Ville Nopri

CANopen-tiedonsiirto-ohjelma Linux-käyttöjärjestelmälle: Metropolian ConceptCar-hanke

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Auto- ja kuljetustekniikka

Insinöörityö

29.5.2013

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Ville Nopri CANopen-tiedonsiirto-ohjelma Linux-käyttöjärjestelmälle: Metropolian ConceptCar-hanke 38 sivua + 4 liitettä 29.5.2013
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Auto- ja kuljetustekniikka
Suuntautumisvaihtoehto	Autosähkötekniikka
Ohjaaja(t)	Projekti-insinööri Harri Santamala Autoelektroniikan lehtori Vesa Linja-aho

Tämä insinöörityö käsitteli CANopen-protokollaperheeseen perustuvan tiedonsiirto-ohjelman ja tiedonsiirtorajapintakirjaston suunnittelua ja toteutusta Linux-käyttöjärjestelmälle. Työn alussa tutustuttiin erilaisiin tapoihin toteuttaa Linuxin prosessien välinen tiedonsiirto sekä CANopen-protokollaperheeseen ja sen ohjelmointikirjastoihin. Työssä selvitettiin aluksi prosessien välisen tiedonsiirron teoriaa ja CANopen-protokollaperheen eri protokollien toimintaa.

Ohjelman ja rajapintakirjaston suunnittelun alussa käytiin läpi käyttötapaukset, joiden perusteella ohjelmisto toteutettiin. Työssä päädyttiin käyttämään prosessien välistä D-Bus-tiedonsiirtomekanismia ja QtDBus-moduulia. CANopen-kirjastoista valittiin käytettäväksi Metropolian koneautomaationlaboratorion kehittämä Minicanopen-ohjelmointikirjasto.

Toteutettu ohjelmisto sisältää CANopendbus-palvelinohjelman, joka toimii Linux-käyttöjärjestelmän taustaohjelmana. Ohjelma lukee ja kirjoittaa dataa CAN-väylälle käyttäen useita CANopen-protokollia. Käyttöjärjestelmän muut ohjelmat ovat yhteydessä Canopendbus-ohjelmaan Linuxin prosessien välisen D-Bus-tiedonsiirtomekanismin avulla.

Toteutettu rajapintakirjasto on yksinkertainen, ja sen avulla voidaan, C++-kielen lisäksi käyttää QML-kieltä CAN-väylän datan lukemiseen ja kirjoittamiseen. Työssä toteutettua ohjelmaa ja rajapintaa tullaan käyttämään Metropolian ConceptCar-hankkeessa syntyvässä kompaktin kaupunkiauton In-Vehicle Infotainment -järjestelmässä. Ohjelman ja rajapinnan toteutuksessa päästiin haluttuihin tavoitteisiin. Ohjelman ja rajapinnan kehitys jatkuu tämän työn jälkeen yhdessä korisähköjärjestelmän kehityksen kanssa.

Avainsanat	CANopen, IPC, D-Bus, Qt, C++, QML, IVI

Author(s) Title Number of Pages Date	Ville Nopri Design of CANopen Communication Program for Linux Operating System: Metropolia ConceptCar Project 38 pages + 4 appendices 29 May 2013
Degree	Bachelor of Engineering
Degree Programme	Automotive and Transport Engineering
Specialisation option	Automotive Electronics Engineering
Instructor(s)	Harri Santamala, Project Engineer Vesa Linja-aho, Senior Lecturer in Automotive Electronics

The aim of this Bachelor's Thesis was to design and implement CANopen protocol stack based communication program and application programming interface (API) for Linux operating system. At the beginning of the thesis different methods to implement the Inter-process communication to Linux operating system were examined and CANopen protocol stack programming libraries were studied.

Software use cases were defined first and later they were used in the software development. A decision was made to use D-Bus and QtDBus-module for the Inter-process communication. Minicanopen-library was also used for CANopen protocol stack library. It was developed by the Machine Automation Laboratory of Helsinki Metropolia University of Applied Sciences.

The implemented software includes a server-program CANopendbus, which is running in the Linux operating system. The program can read and write data via CAN-bus using multiple CANopen protocols. User Interface applications of the operating system are connected to the server-program via D-Bus Inter-process communication method.

The application programming interface is simple and it can be used with C++- and QML-programming languages to read and write data via CAN bus. The server-program and API will be used in the In-Vehicle Infotainment system of the ConceptCar project. The desired objective was achieved. The server-program and API development continues with the development of the ConceptCar's electric-system.

Keywords	CANopen, IPC, D-Bus, Qt, C++, QML, IVI

Sisällys

Lyhenteet

API	*Application programming interface*. Ohjelmointirajapinta, jonka välityksellä ohjelmat voivat keskustella keskenään.
CAN	*Controller Area Network.* Sarjamuotoinen verkko.
COBID	*Communication Object Identifier*. Sama kuin CAN id, mutta sisältää myös tiedon CANopen viestityypistä.
IPC	*Inter-process communication.* Prosessien välinen tiedonsiirto.
NMT	*Network management*. CANopen-verkonhallintaprotokolla.
PDO	*Process data object*. Prosessiviesti objekti, CANopen PDO -protokolla.
QML	*Qt Meta* Language. Qt Ohjelmointikieli graafiseen käyttöliittymäkehitykseen.
SDO	*Service data object*. CANopen SDO -protokolla laitteiden väliseen konfiguroimiseen.
XML	*Extensible Markup Language*. Merkintäkieli, joka määrittelee säännöt XML-tyyppiselle tiedostoformaatille.

1 Johdanto

Tämän työn tarkoituksena oli suunnitella ja ohjelmoida tiedonsiirto-ohjelma kaupunkiautoprojektiin osana In-Vehicle Infotainment -järjestelmää. Tiedonsiirto-ohjelman tehtävä on lukea dataa CAN-väylältä ja siirtää sitä prosessien välisellä tiedonsiirtomekanismilla muille käyttöjärjestelmän ohjelmille sekä kirjoittaa CAN-väylälle dataa prosessien välisellä tiedonsiirtomekanismilla.

Työ toteutetaan Metropolia Ammattikorkeakoulun ConceptCar-hankkeelle. ConceptCar-hankkeessa on tarkoitus suunnitella ja toteuttaa kompakti kaupunkiauto. ConceptCar-hanke on Tekes-rahoitteinen projekti. Auto esitellään 2014 Geneven autonäyttelyssä.

Työn alussa käydään läpi erilaisia vaihtoehtoisia toteutuksia prosessien väliselle tiedonsiirrolle. Seuraavaksi esitellään CANopen-ohjelmointikirjasto, Linuxin D-Bus IPC (Inter-process communication) -mekanismi sekä ohjelman suunnitteluvaiheet ja toteutus. Lopuksi tarkastellaan työn onnistumista ja jatkokehitysmahdollisuuksia.

2 Työn tausta ja rajaus

2.1 In-Vehicle Infotainment

IVI eli In-Vehicle Infotainment järjestelmä on autoon integroitu viihde- ja informaatiojärjestelmä, joka sisältää yleensä auton informaatio-, toimilaitteiden ohjaus-, multimedia-, navigointi- ja internetominaisuudet. Autoissa on ollut jo pidemmän aikaa jonkinlaisia IVI-järjestelmiä. Kuitenkin vasta viimeaikoina järjestelmistä on alettu kehittää näyttävämpiä ja ominaisuuksiltaan kattavampia. Yksi esimerkki uudemman sukupolven IVI-järjestelmästä on Tesla Model S -auto, jossa on 17”:n kosketusnäyttö keskikonsolissa ja 12,3”:n näyttö mittaristossa. Teslassa myös useimpien toimilaitteiden ohjaukset on toteutettu IVI-järjestelmän kosketusnäytön avulla, mikä vähentää perinteisiä sähkökytkimiä. (1; 2.)

Kaupunkiauton IVI-järjestelmä sisältää kolme näyttöä, joilla jokaisella on tietokone ohjaimena. Näiden tietokoneiden kesken tietoa jaetaan autoon sijoitetulla Ethernet-

verkolla. Näyttöjä käytetään mittaristona, keskikonsolin viihde- ja informaationäyttönä sekä apukuljettajan viihdenäyttönä. Kaupunkiauton IVI-järjestelmän yksi keskeinen tavoite on fyysisten sähkökytkimien vähentäminen auton kojetaulusta. Näiden toimintojen ohjaus on tarkoitus siirtää IVI-järjestelmän kosketusnäytölle. Kaupunkiauton korisähköjärjestelmässä käytetään CAN-väylää erilaisten toimintojen ohjaukseen, joten ainakin yhden tietokoneen täytyy olla yhteydessä CAN-väylään.

2.2 Linux-käyttöjärjestelmänä

Kaupunkiauton tietokoneiden käyttöjärjestelmäksi valittiin Linux. Linuxin suurimpana etuna on sen tuki useille alustoille. Se on hyvin yleisesti käytössä maailmalla sulautetuissa järjestelmissä. Vaikka käyttäjä ei sitä aina huomaakaan, niin Linux-käyttöjärjestelmän voi löytää kaupan maksupäätteistä, omasta matkapuhelimesta, televisiosta tai vaikka mp3-soittimesta. Linuxin tuki mukautua järjestelmän vaatimusten mukaan ratkaisi sen valinnan kaupunkiauton tietokoneiden käyttöjärjestelmäksi.

2.3 CANopen-protokollaperheen valinta

Kaupunkiautossa käytetään CANopen-protokollaperhettä CAN-väylän tiedonsiirrossa. CANopen on korkeamman tason protokolla, joka toteuttaa OSI-mallin verkko-, kuljetus-, istunto- ja esitystapakerroksen. CANopen valittiin kaupunkiauton CAN-verkon protokollaksi siitä syystä, että se on korkeamman tason protokolla ja siitä löytyy erilaisia verkonhallintaominaisuuksia.

2.4 Prosessien välisen tiedonsiirron tarve

Prosessien välistä tiedonsiirtoa tarvitaan esimerkiksi, kun mittaristolle halutaan tulostaa ajoneuvon nopeus CAN-väylältä tai kun käyttäjä haluaa säätää ilmastoinnin puhaltimen nopeutta, jolloin käyttäjän haluama nopeus välitetään tuuletinta ohjaavalle laitteelle. Näiden ominaisuuksien mahdollistamiseen ohjelmassa käytetään prosessien välistä tiedonsiirtoa käyttöliittymän ohjelman ja CAN-väylän välillä. Toteutukseen on olemassa useita eri tapoja ja niitä tarkastellaan luvussa 3.

3 Prosessien välinen tiedonsiirto

3.1 Yleistä prossien välisestä tiedonsiirrosta

Linuxissa kuten myös muissa käyttöjärjestelmissä prosessien eli ohjelmien välisellä tiedonsiirrolla (IPC) tarkoitetaan datan siirtoa prosessien välillä (1, s. 95). Prosessien väliseen tiedonsiirtoon Linuxissa on useita tapoja, joista yleisimmät ovat

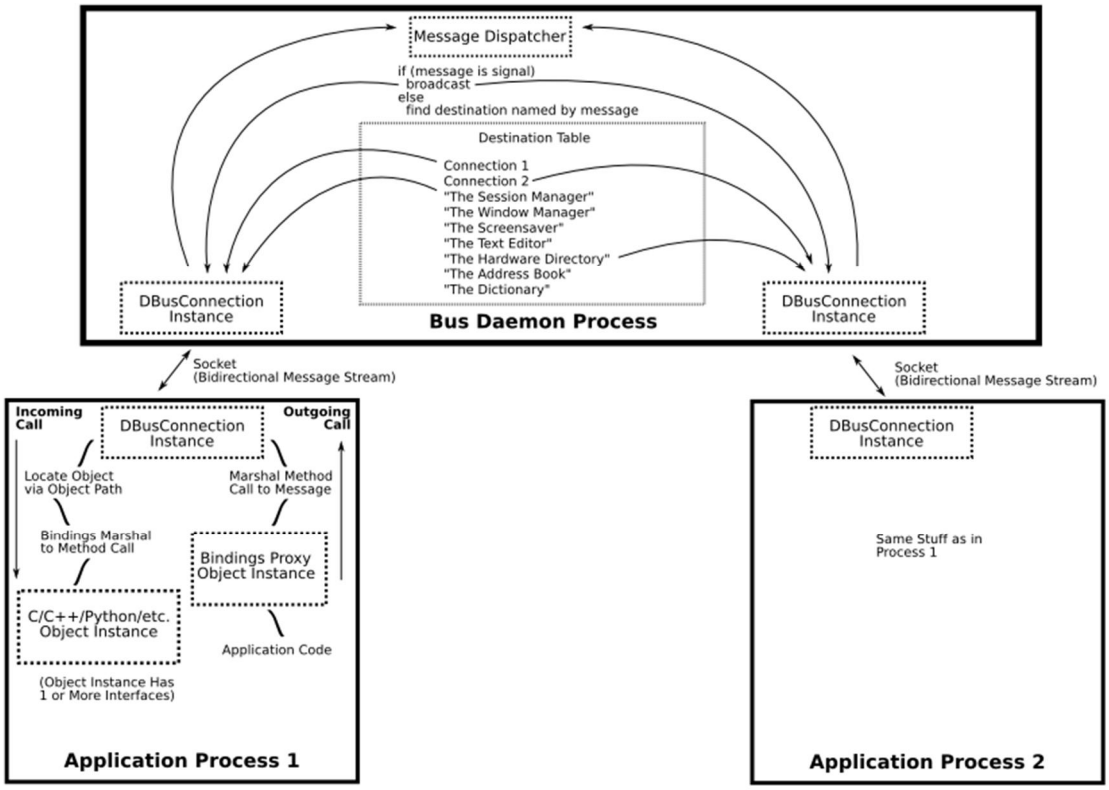
- Shared memory eli jaettu muisti jossa prosessit lukevat ja kirjoittavat dataa tiettyyn yhteiseen RAM muistialueeseen (1, s. 96)
- Mapped memory eli kartoitettu muisti joka toimii samalla tavalla kuin jaettu muisti, mutta tässä käytetään tietokoneen massamuistia jossa voi olla jaettu tiedosto tai tiedostoja tiedostojärjestelmässä (1, s. 96)
- Pipes eli putkitus, jossa edellisen prosessin ulostulovirta ohjataan seuraavalla prosessille (1, s. 96)
- FIFO(first-in first-out), joka on samankaltainen kuin putkitus tai mapped memory, mutta tässä tapauksessa prosessit käyttävät tiedostoa, johon on toteutettu FIFO-tyyppinen puskuri (1, s. 96)
- Socket on IPC-mekanismi, jossa luodaan yhteys kahden socket:n välille ja näiden välillä voidaan siirtää dataa (1, s. 96)
- D-Bus on viestiväylätyyppinen IPC-mekanismi, jossa on siirrettävän viestin lisäksi lähettäjän nimi, vastaanottajan nimi, viestin tyyppi ja itse viesti. D-Busilla voidaan myös lähettää viestejä yhdeltä prosessilta monelle samaan aikaan. D-Busille löytyy myös useita korkeamman tason ohjelmointirajapintoja. (4.)

3.2 D-Bus

D-Bus on suunniteltu Linux-käyttöjärjestelmän prosessien väliseen tiedonsiirtoon. D-Bus on nopea ja kevyt tiedonsiirtomekanismi, sekä siinä on alhainen viive ja pieni overhead. Pieni overhead tarkoittaa yleisesti sitä, että viestikehyksen hyötysuhde on hyvä ja viestikehyksessä on enemmän hyötykuormabittejä kuin esimerkiksi identifiointi-, tarkistus- ja synkronointibittejä. D-Bus on yhteyspohjainen tiedonsiirtomekanismi, mikä tarkoittaa sitä, että siirrettäessä dataa D-Busin välityksellä, prosessien täytyy ensiksi muodostaa yhteys ja toteuttaa ns. kättely ennen varsinaista datansiirtoa. D-Bus:n välityksellä dataa voi siirtää joko synkronisesti tai asynkronisesti. D-Busia voi käyttää useilla eri ohjelmointikielillä kuten C, C++, Java, Perl ja Python kielillä. (5; 6.)

D-Bus-viestit välitetään Linuxissa pyörivälle taustaprosessille D-Bus Daemon. D-Bus Daemonia käytetään D-Bus-ohjelmointirajapinnan (API) välityksellä, joka sisältää metodit D-Busin yhteyden luontiin ja viestien välitykseen. D-Bus Daemon taas on ohjelma joka reitittää D-Busin viestit oikeaan osoitteeseen ohjelmien välillä. (4.)

D-Busia käyttävä ohjelma luo aina DBusConnection-instanssin eli olion D-Bus API:n avulla. Kuvasta 1 nähdään, että molemmat D-Busia käyttävät ohjelmat ovat luoneet DBusConnection olion. Tämän olion välityksellä ohjelmat ovat yhteydessä D-Bus Daemon -ohjelmaan ja voivat lähettää sen kautta viestejä toisilleen. D-Bus Daemon -ohjelmassa on Message Dispatcher eli viestien välittäjämoduuli. Tämä moduuli tarkistaa, onko viesti signaali vai joku muu. Jos viesti on signaali, se lähetetään broadcast-tyyppisesti eteenpäin kaikille ohjelmille, jotka ovat yhteydessä D-Bus Daemon -ohjelmaan. Jos viesti on tarkoitettu tietylle ohjelmalle, sen osoite tarkistetaan Destination Tablesta ja ohjataan oikeaan osoitteeseen. (2.)



Kuva 1. Konseptikuva D-Bus tiedonsiirrosta kahden ohjelman välillä (4).

D-Bus sisältää kaksi eri väylää. System bus on tarkoitettu käytettäväksi käyttöjärjestelmän kaikkien ohjelmien kesken. Tätä käytetään esimerkiksi, kun välitetään käyttöjärjestelmän tapahtumia eteenpäin, kuten uuden laitteen kytkeytyminen tietokoneeseen.

Toinen väylätyyppi on session bus, jota käyttävät käyttäjän ohjelmat tai kirjautumistapahtuman käyttäjät. Esimerkiksi kun käyttäjä vaihtaa kappaletta musiikkisoittimessa, tämä tieto välitetään usein session bus -väylää pitkin. (6; 7.)

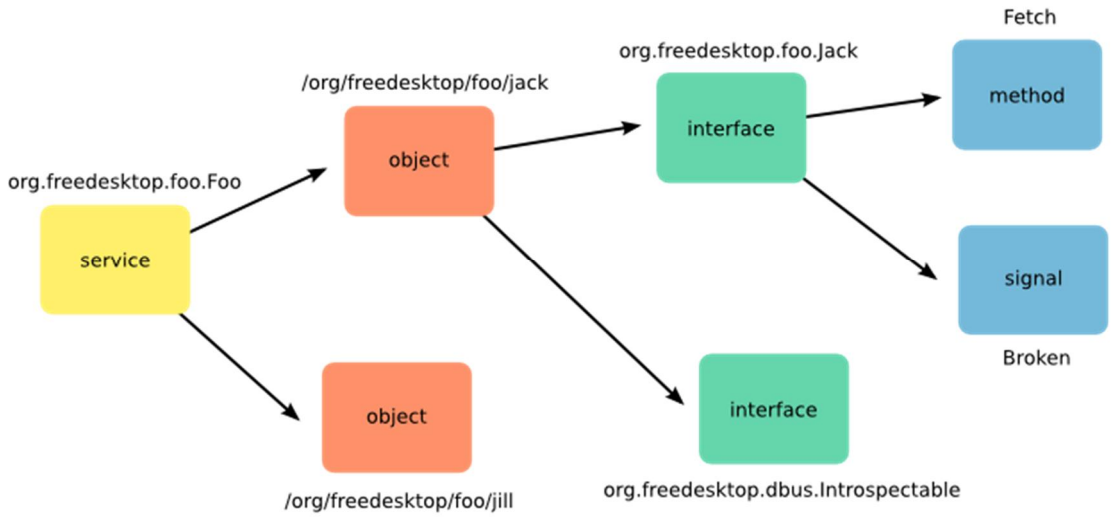
Kun jokin ohjelma ottaa yhteyden D-Bus Daemon -ohjelmaan, sille täytyy antaa jokin osoite, jolla muut ohjelmat voivat olla yhteydessä siihen. Esimerkiksi tekstieditorin käyttämä osoite voisi olla org.freedesktop.TextEditor. Tämäntyyppisiä selkokielelisiä osoitteita kutsutaan nimellä well-known names. Selkokielisestä osoitteesta puhutaan joissakin dokumenteissa myös nimellä service name. D-Bus Daemon antaa vielä oman uniikin nimensä ohjelmalle, joka alkaa kaksoispisteellä ja jonka jälkeen tulee uniikki merkkijono, joka ei tarkoita mitään, esimerkiksi :34-907. Tätä osoitetta kutsutaan nimellä unique connection name. (4; 7.)

D-Bus tarjoaa konseptin nimeltä object path. Tämän avulla voidaan käyttää toisen ohjelman objekteja eli olioita D-Busin välityksellä. Olioille annetaan myös nimi kuten D-Bus-palvelulle. Esimerkiksi tekstieditorilla voisi olla olio clipboard, jonka osoite olisi /org/freedesktop/texteditor/clipboard. Olioille annetaan siis jonkin polun nimi, mistä ne löytyvät. (4.)

Jokaisella oliolla voi olla kahta eri tyyppiä olevia jäseniä, metodeita tai signaaleita. Metodit ovat operaatioita joita voi kutsua D-Busin välityksellä. Niillä voi olla parametreja ja ne voivat palauttaa paluuarvoja. Metodit ovat käytännössä ohjelmassa olevia funktioita, jotka on määritelty D-Bus-spesifikaation mukaan. Signaalit ovat nimensä mukaisesti signaaleita, joita yleensä käytetään tapahtumapohjaiseen tiedonsiirtoon. Signaalit voivat sisältää myös parametreja. (4.)

Jokaisella oliolla on yksi tai useampi rajapinta. Rajapintaan on määritelty olion metodit ja signaalit, joita se tarjoaa. Esimerkiksi tekstieditorilla voisi olla metodi org.freedesktop.TextEditor.clipboard.load, joka palauttaisi leikepöydän sisällön. (4.)

Kuvassa 2 on esimerkki D-Bus-spesifikaation mukaisista palveluiden, olioiden, rajapintojen, signaalien ja metodien nimeämisestä ja yhteydestä. Kuvassa 2 on kuvattu keltaisella palvelu, oranssilla oliot, vihreällä rajapinnat, sinisellä metodit ja signaalit.



Kuva 2. D-Busin toiminnallisuuksia ja nimeämisiä havainnollistava kuva (7).

D-Bus-prosessien välinen tiedonsiirtomekanismi valittiin useista syistä:

- D-Bus tiedonsiirron viive on pieni, joten tiedonsiirto on riittävän reaaliaikaista.
- CAN-väylän ja D-Busin välillä ei siirretä paljoa dataa, joten D-Bus Daemon ei vie liikaa prosessoriaikaa.
- D-Busiin löytyy useita korkeamman tason ohjelmointirajapintoja, joita on helppo käyttää.
- Qt-ohjelmointikirjasto sisältää D-Bus-moduulin QtDBus, joka on korkeamman tason ohjelmointirajapinta. IVI-järjestelmän käyttöliittymän ohjelmien kehityksessä käytetään Qt:ta, niin kyseinen QtDBus-moduuli on suoraan yhteensopiva näiden ohjelmien kanssa.

3.3 QtDBus

Qt-ohjelmointikirjasto sisältää korkeamman tason D-Bus-ohjelmointikirjaston (API) QtDBus. QtDBus-moduuli sisältää 18 luokkaa, joilla voidaan käyttää D-Busia. Yksi tärkeimmistä luokista on QBusConnection, jonka avulla ohjelma on yhteydessä D-Bus Daemon -ohjelmaan ja jonka avulla alustetaan D-Bus-yhteys. (8; 9.)

3.3.1 QDBusAbstractAdaptor

QDBusAbstractAdaptor on abstrakti luokka. QDBusAbstractAdaptor tarjoaa rajapinnan olioille, joiden halutaan olevan yhteydessä D-Busin välityksellä järjestelmän muihin ohjelmiin. Tämä rajapintaluokka luodaan perimällä QDBusAbstractAdaptor-luokan kyseisen luokan kantaluokaksi. Tämä rajapintaluokka yleensä toimii vain välittäjänä esimerkiksi QObject-luokasta periytyvän luokan julkisille funktioille, slot-funktioille ja signaaleille. QDBusAbstractAdaptor-luokasta perityn luokan täytyy toteuttaa Q_CLASSINFO-makro sen määrittelyssä. Q_CLASSINFO-makrolle annetaan parametrina nimi sekä D-Bus-rajapinnan nimi. (10.) Esimerkkikoodissa 1 on havainnollistettu tätä määrittelyä.

```
Q_CLASSINFO("D-Bus Interface", "com.canbus.Data")
```

Esimerkkikoodi 1. Qt luokkainformaatiomakron määrittely.

QDBusAbstractAdaptor-luokasta periytyvälle luokalle täytyy varata muisti keosta, eikä sitä saa itse vapauttaa. Muisti vapautuu automaattisesti, kun QDBusAbstractAdaptor-tyyppisen olion isäntäolio tuhotaan. (10.)

QDBusAbstractAdaptor-luokkaa ei toteuteta tässä työssä manuaalisesti vaan tässä työssä käytetään Qt:n tarjoamia työkaluja, joilla voidaan automatisoida koodin generointia. Näistä työkaluista kerrotaan ohjelman suunnittelussa ja toteutuksessa luvussa 6.1.2.

3.3.2 QDBusAbstractInterface

QDBusAbstractInterface on myös abstrakti luokka, joka tarjoaa yhteyden D-Bus-rajapinnan palveluihin. Kun QDBusAbstractAdaptor tarjoaa rajapinnan tietyn olion palveluihin, QDBusAbstractInterface tarjoaa yhteyden D-Bus-rajapinnan palveluihin. D-Bus-rajapinta voi sisältää esimerkiksi useamman adaptor-olion. (11.)

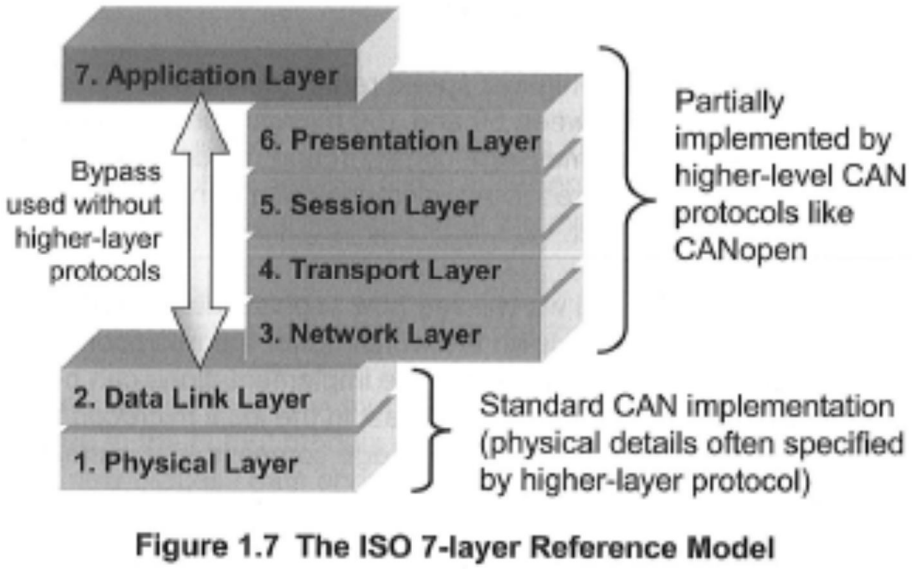
Myöskään tätä luokkaa ei toteuteta tässä ohjelmassa manuaalisesti vaan tässä käytetään samoja Qt:n tarjoamia työkaluja, jotka luovat valmiin QDBusAbstractInterface D-Bus -rajapinnan.

4 CANopen-protokollaperhe

Tässä luvussa käsitellään pintapuolisesti, mitä CANopen-protokollaperhe sisältää, sekä tutustutaan niihin CANopen-protokolliin, joita tarvitaan myöhemmin tässä työssä suunniteltaessa ja toteutettaessa tiedonsiirto-ohjelmaa.

CANopen toteuttaa OSI-mallin kerrokset 3–6 (kuva 3). Se on siis korkeamman tason protokolla, kun taas CAN-standardi määrittelee ainoastaan OSI-mallin fyysisenkerroksen ja siirtokerroksen. CANopen ei ole yksittäinen protokolla vaan TCP/IP:n tavoin protokollaperhe, joka sisältää useita protokollia. (12, s. 1; 13.)

CANopen-protokollaperheen käyttö on laajaa; se käsittää alueita kuljetuksesta, terveydenhuollosta, viihdekäyttöön ja tiedesovelluksiin. CANopen-protokollaperhettä käytetään myös henkilöautopuolella paljon eri järjestelmissä kuten ESP-järjestelmissä ja hybridijärjestelmissä. (14.)



Kuva 3. OSI-malli, josta näkyy, mitkä kerrokset CANopen-protokollaperhe toteuttaa (15, s. 18).

4.1 Objektikirjasto

Objektikirjasto toimii CANopen-solmun rajapintana ja tietovarastona. Objektikirjastoon tallennetaan CANopen-liikennöintiin tarvittavia parametreja sekä solmujen välillä siirrettävää tietoa. Kaikilla solmuilla on oma objektikirjastonsa. Objektikirjasto sisältää 65535 indeksiä, ja jokainen indeksi on jaettu ali-indeksiin, joita voi olla maksimissaan 254 kappaletta. (12, s. 7.)

4.2 SDO-protokolla

SDO-protokolla (Service Data Object) on kehitetty tiedonsiirtoon, sillä voidaan lukea ja kirjoittaa objektikirjastoon. SDO-protokolla on raskas tapa lähettää prosessiviestejä. Tähän on kehitetty kevyempi vaihtoehto, PDO-protokolla, josta kerrotaan seuraavassa luvussa. Tavallisesti SDO-protokollan mukaista tiedonsiirtoa voidaan suorittaa vain CANopen-isäntäsolmun ja CANopen-orjasolmun välityksellä. (12, s. 8.)

4.3 PDO-protokolla

PDO-protokolla (Process Data Object) sisältää säännöt, joilla voidaan lähettää tai vastaanottaa prosessisignaaleita. PDO-viestejä on kahta tyyppiä: TPDO (Transmit Process Data Object) ja RPDO (Receive Process Data Object). TPDO lähettää PDO-viestejä solmulta verkkoon ja RPDO taas vastaanottaa PDO-viestejä verkosta solmuun. PDO-viestit sisältävät useita ominaisuuksia, kuten COBID-osoitteen, muuttujatyypin, datan koon ja itse siirrettävän datan. Signaalin arvo kirjoitetaan objektikirjastoon, jonka kautta ohjelma voi kirjoittaa tai lukea PDO-tietoa. PDO-viestien lähetys voidaan toteuttaa synkronisesti, asynkronisesti, tietystä liipaisusta tai näiden yhdistelmänä. (12, s. 9.)

4.4 NMT-protokolla

NMT-protokollan (Network Management) avulla hallitaan CANopen-verkkoa. NMT-isäntä voi esimerkiksi käynnistää ja sammuttaa solmuja verkossa. NMT-protokolla sisältää useita tiloja. Reset node -tilassa toteutetaan kaikki solmun alustukset, paitsi muiden CANopen-protokollien alustuksia ei vielä tehdä. Reset communication -tilassa

solmun CANopen-protokollat alustetaan. Tässä tilassa myös lähetetään boot viesti. Pre-operational-tilassa solmu jää odottamaan NMT-isännän käynnistyslupaa siirtyä Operational-tilaan. Operational-tilassa CANopen-verkko ja -solmu ovat normaalissa toimintatilassa. Tässä tilassa solmut voivat lähettää ja vastaanottaa viestejä. Stopped-tilassa solmu ei pysty vastaanottamaan muita kuin NMT-komentoja ja lähettämään ainoastaan Nodeguarding- tai Heartbeat-protokollan mukaisia viestejä. Tähän tilaan solmu ajautuu yleensä vakavan virheen seurauksena. (12, s. 9.)

4.5 Heartbeat- ja Nodeguarding-protokolla

Nodeguarding-protokolla on kehitetty CANopen-solmujen valvontaan. CANopen-isäntä valvoo tämän protokollan avulla CANopen-solmuja. Nodeguarding-protokolla on vanhentunut, ja se on korvattu Heartbeat-protokollalla. (12, s. 8.)

Heartbeat-protokollan avulla solmu voi lähettää tilatietoa itsestään verkkoon muille solmuille. NMT-isäntä tai muut solmut voivat valvoa kyseistä tilatietoa lähettävää solmua. Heartbeat-protokollaa suositellaan käytettäväksi, koska se vie vähemmän kaistaa. (12, s. 8.)

4.6 Laite- ja sovellusprofiilit

Laiteprofiili (Device profile) on jollekin tietylle laitteelle yksityiskohtaisesti määritelty profiili. Laiteprofiiliin määritellään tietyn laitteen yksityiskohtaisia tietoja. Esimerksi DS-401-laiteprofiili on geneerisille I/O-moduuleille tarkoitettu laiteprofiili. DS-401-laiteprofiiliin on esimerkiksi määritelty analogiatulon näytteenottotaajuus, resoluutio ja mahdolliset liipaisumääritykset. Laiteprofiileita on useita, ja ne on kehitetty tietyn laitteen määrittelyyn. Mittalaitteille, antureille ja akkulatureille on kehitetty omat laiteprofiilit. Sovellusprofiili (Application profile) on samankaltainen profiili kuin laiteprofiili, mutta se on määritelty tietylle sovellukselle, esimerkiksi CiA 417-profiili on määritelty hissijärjestelmän sovellusprofiiliksi. (16.)

4.7 CANopen-ohjelmointikirjastot

CANopen-protokollaperheen käyttöön on olemassa useita kaupallisia sekä avoimeen lähdekoodiin perustuvia ohjelmointikirjastoja. Näissä kirjastoissa on yleensä toteutettu suurin osa CANopen-protokollista. Tässä työssä haluttiin käyttää avoimen lähdekoodin ohjelmointikirjastoa. Toteutettava ohjelma tulee olemaan avoimen lähdekoodin ohjelma, joten ohjelmassa käytettävän CANopen-ohjelmointikirjastonkin kannattaa olla avointa lähdekoodia.

4.7.1 CanFestival

CanFestival on ilmainen avoimeen lähdekoodiin perustuva CANopen-ohjelmointikirjasto. CanFestival-kirjasto on kirjoitettu ANSI-C:llä ja se voidaan kääntää usealle eri arkkitehtuurille, kuten x86-PC:t, mikrokontrollerit ja erilaiset reaaliaikakäyttöjärjestelmät (RTOS). CanFestival-kirjaston runtime-koodi on lisensoitu LGPLv2-lisenssillä ja muut kehitystyökalut, jotka liittyvät siihen, on lisensoitu GPLv2-lisenssillä. CanFestival sisältää suurimman osan CANopen-protokollista, sekä siinä on valmiina muutamia laiteprofiileita. CanFestival-ohjelmointiympäristöön löytyy useita työkaluja, kuten objektikirjastoeditori, virtuaalinen DS-401-IO-laite ja CANopen-shell-hallintaohjelma. (17; 18, s. 42.)

Objektikirjastoeditorin avulla voidaan määritellä CANopen-solmun objektikirjaston parametrit. Sillä voidaan määritellä sekä orja- että isäntä-solmuja. Määrittelyn jälkeen ohjelma osaa automaattisesti generoida C-lähdekoodin näistä tiedoista. Generoitu ja mahdollisesti muokattu lähdekoodi käännetään lopuksi suoritettavaksi ohjelmaksi. Näin vähenee käsin kirjoitettavan koodin määrä.

4.7.2 Minicanopen

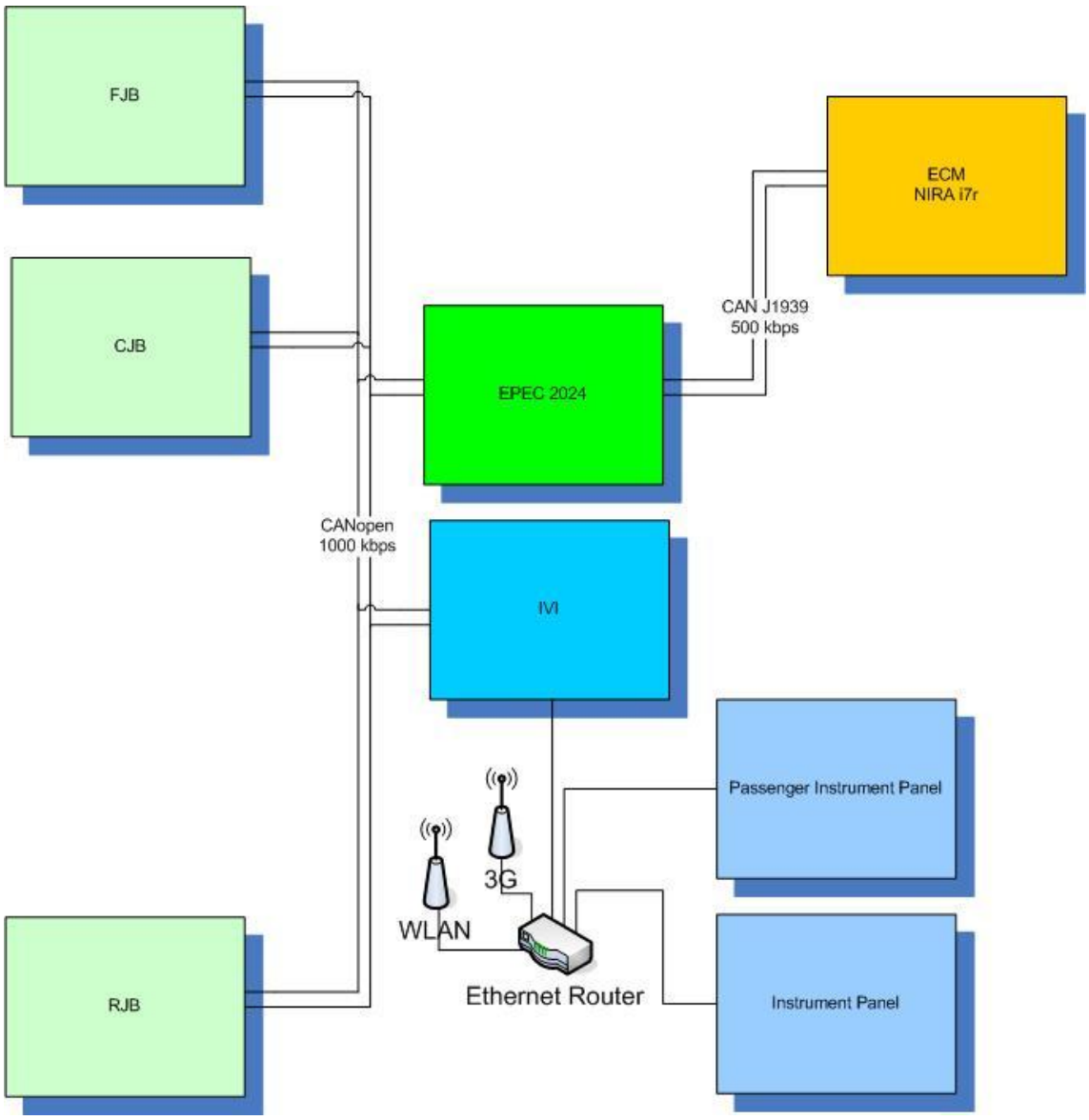
Minicanopen on Metropolian koneautomaationlaboratorion kehittämä kirjasto, jolla voidaan käyttää useimpia CANopen-protokollia. Kirjasto on kirjoitettu C++-kielellä, ja se on toteutettu CiA 301-spesifikaation mukaan. Kirjasto on osittain keskeneräinen; esimerkiksi objektikirjastoa ei ole toteutettu loppuun asti.

Minicanopen valittiin tähän ohjelmaan sen takia, että sitä oli aiemmin käytetty jo onnistuneesti muissa Metropolian projekteissa sekä se on kohtalaisen yksinkertainen käyttää.

5 Kaupunkiauton sähköjärjestelmä

IVI-järjestelmän suunnittelussa mietin erilaisia vaihtoehtoja tiedonsiirron toteuttamisessa korielektroniikan väylältä IVI-järjestelmälle ja päädyttiin kuvan 4 mukaiseen tiedonsiirron toteuttamiseen. IVI-tietokoneen tarvitsee kuitenkin olla yhteydessä mittariston ja apukuljettajan tietokoneisiin, joten oli luonnollista, että näille tarvittava tiedonsiirto korielektroniikan CAN-väylältä tuodaan IVI-tietokoneelta.

Kuvassa 4 havainnollistetaan kaupunkiauton sähköjärjestelmää. Front Junction Box (FJB), Central Junction Box (CJB) ja Rear Junction Box (RJB) ovat korielektroniikan CANopen-orjasolmuja. Niihin on kytketty lähes kaikki korielektroniikan vaatimat sähköiset toimilaitteet ja tunnistimet. Ne ovat CAN-väylällä yhteydessä EPEC 2024 PLC -ohjainlaitteeseen, joka toimii CANopen-master-ohjainlaitteena korielektroniikan CAN-verkossa. EPEC-ohjainlaite on myös yhteydessä NIRA-dieselmoottorinohjainlaitteeseen ja välittää tietoa sen CAN-väylästä korielektroniikan CAN-väylälle. IVI-tietokone on yhteydessä korielektroniikan CAN-väylään. Mittariston- ja apukuljettajantietokoneet ovat yhteydessä Ethernet-verkon välityksellä IVIin.



Kuva 4. Järjestelmäkaavio kaupunkiauton sähköjärjestelmästä.

6 Ohjelman suunnittelu ja toteutus

Tässä kappaleessa tarkastellaan D-Bus-palvelinohjelman ja asiakasrajapinnan suunnittelua ja toteuttamista. Alussa käydään läpi asioita, joita ohjelmalta ja rajapinnalta vaaditaan sekä myöhemmin ohjelman ja rajapinnan toteuttamista.

Työkaluina tässä työssä käytetään seuraavia työkaluja:

- ohjelmointikielenä C++-kieltä
- Qt-kirjastoja sekä erityisesti QtDBus-moduulia
- ohjelmointiympäristönä (IDE) Qt Creatoria
- käyttöjärjestelmänä uusinta vakaata Ubuntu Linux -jakelua.

6.1 Palvelinohjelman vaatimusmäärittely

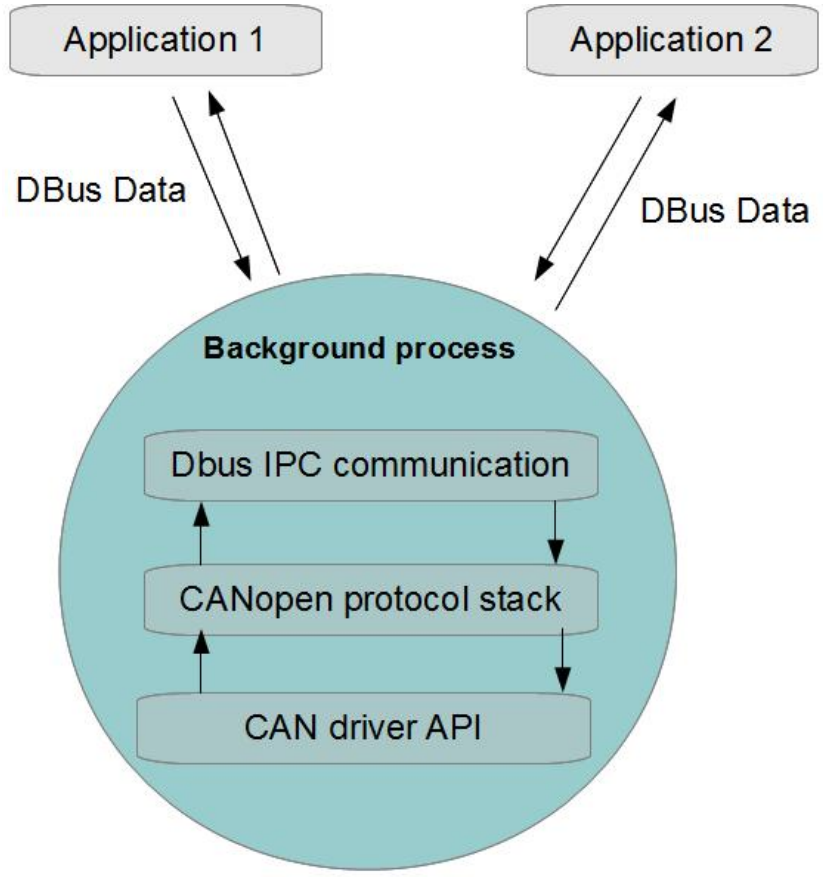
Palvelinohjelman tehtävä on lukea ja kirjoittaa CAN-väylälle, välittää dataa Dbusissa sekä luettua dataa UDP-protokollaa käyttäen Ethernet-verkkoon. Palvelinohjelman on tarkoitus olla taustaohjelma Linuxissa ja sen on toimittava täysin automaattisesti. Ohjelman on tarkoitus käynnistyä automaattisesti Linuxin käynnistyessä, sekä lisäksi ohjelman kaatuessa sen on käynnistyttävä automaattisesti uudelleen. Kun CAN-väylää lukeva ohjelma toteutetaan omaksi ohjelmakseen, niin se vähentää todennäköisyyttä käyttöliittymän vikatilanteeseen, joka näkyisi taas suoraan käyttäjälle. Jos palvelinohjelmaan tulisi vakava vika, pysyisi käyttöliittymä edelleen toiminnassa ja käyttäjälle voitaisiin vain ilmoittaa viasta sekä käynnistää palvelinohjelma uudelleen. Palvelinohjelmaa tullaan käyttämään ohjelmointirajapinnan kautta. Ohjelmointirajapinnan suunnittelusta ja toteutuksesta kerrotaan luvussa 6.2.

Palvelinohjelmalta vaaditaan seuraavia ominaisuuksia:

- Ohjelmaa pitää pystyä käyttämään rajapinnan välityksellä monta asiakasohjelmaa yhtä aikaa.
- Ohjelman on tunnistettava, kun siihen on rekisteröitynyt asiakasohjelma D-Bus:n välityksellä.
- Ohjelman on kyettävä käsittelemään CANopen-paketit riittävän nopeasti, jotta viestejä ei häviä CAN-laitteen puskurista.

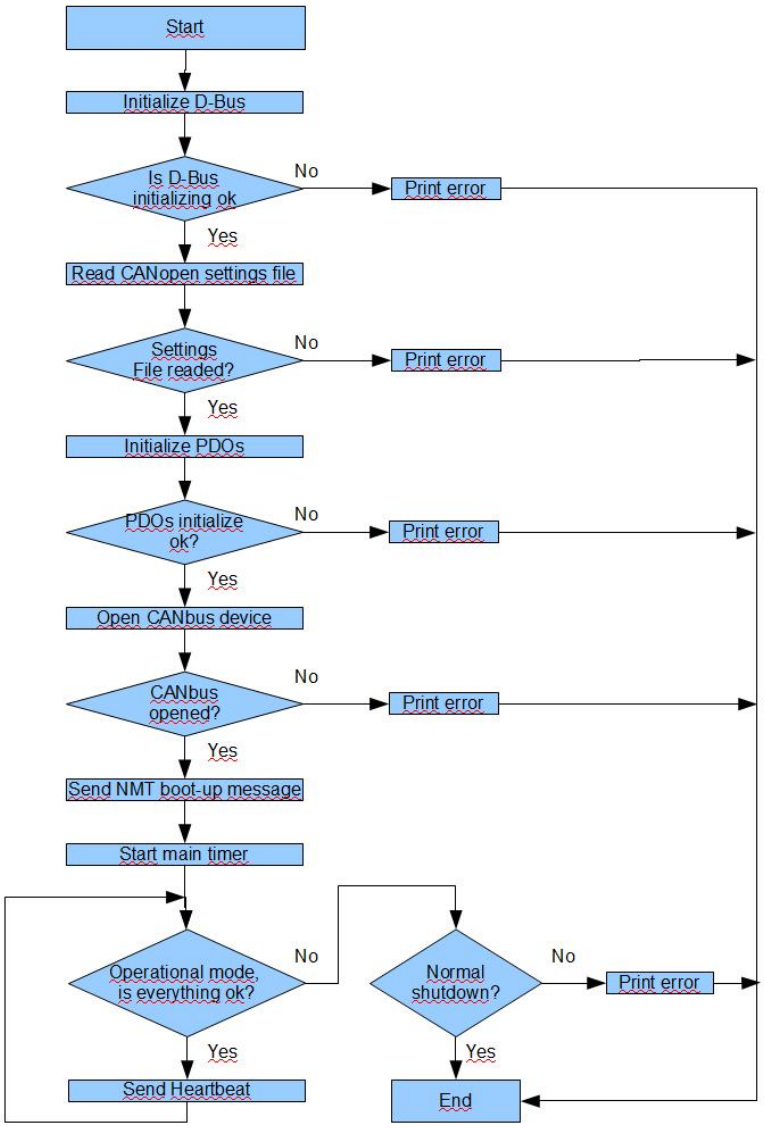
- Ohjelmaan määritetään ajastimet minimi- ja maksimiajalle, kun lähetetään CAN-väylältä saapuneita viestejä D-Busille.
- TPDO- ja RPDO-viestien määrittelyt täytyy pystyä lukemaan tiedostoista ohjelman käynnistyessä, jotta niitä on helppo lisätä tai muokata myös tulevaisuudessa.
- Ohjelman on lähetettävä CAN-väylältä luettu data myös Ethernet-verkkoon riittävän reaaliaikaisesti.

Kuvassa 5 on havainnollistettu ohjelman rakentumista eri moduuleista. Alimpana on CAN-ajurin ohjelmointirajapinta, joka on yhdistetty Minicanopen-kirjastoon. Keskimmäisenä on Minicanopen-kirjasto ja ylimmäisenä D-Bus IPC -tiedonsiirtomekanismi.



Kuva 5. Kaavio taustaohjelman tiedonsiirrosta asiakasohjelmille.

Kuvan 6 vuokaaviossa on käyty läpi ohjelman käynnistymisen päävaiheet. Vuokaavio ei ole tarkka seloste ohjelman toiminnasta, eikä se ota kantaa miten viestit kulkevat D-Busin läpi. Vuokaavion alussa lähdetään alustamaan ohjelman eri vaiheita. Ensiksi alustetaan D-Bus, CANopen, CAN-ajuri ja lopuksi käynnistetään pääajastin. Jos alustuksessa tulee virhe, tulostetaan se ennen ohjelman lopettamista.



Kuva 6. Palvelinohjelman käynnistysvaiheista suunnitteluvaiheessa piirretty vuokaavio.

6.1.1 D-Bus-tiedonsiirto

D-Bus-tiedonsiirrossa käytetään jo aikaisemmin mainittua Qt-kirjaston QtDBus-moduulia. Ohjelman main-funktiossa alustetaan D-Bus-tiedonsiirto ja varataan muistia tarvittaville olioille. DbusHandler-luokka toimii välittäjänä CANopen-tiedonsiirron ja D-Bus-tiedonsiirron välillä. DbusHandler-luokka muodostaa CanOpenHandler-luokasta olion, mitä kautta sillä on yhteys CANopen-dataan. DbusHandler-luokalla on myös viite ComCanbusDataInterface-olioon, mistä taas on yhteys D-Bus-tiedonsiirtoon. Esimerkkikoodissa 2 on havainnollistettu ohjelman main-funktiossa D-Busin alustusta.

DbusHandler-luokasta muodostetaan olio ohjelman main-funktiossa, jossa oliolle annetaan parametrina viite ComCanbusDataInterface-luokkaan. ComCanbusDataInterface-luokka on peritty QDBusAbstractInterface-luokasta ja generoitu automaattisesti D-Bus-työkalujen avulla. D-Bus-koodin generoinnista kerrotaan enemmän seuraavassa kappaleessa. Toisena parametrina DbusHandler-oliolle annetaan QCoreApplication-olion osoite, joka on myös muodostettu main-funktiossa.

CAN-väylään ja CANopen-ohjelmointikirjastoon liittyvät alustukset tehdään vasta tämän jälkeen. Jos D-Bus-alustuksessa tulee jokin virhe, ei ohjelman käynnistystä kannata jatkaa, sekä jos myöhemmin CAN-väylään liittyvissä alustuksissa tulee virheitä, voidaan tieto näistä virheistä välittämään asiakasohjelmille D-Busin kautta.

```
CanbusData *pCanbusData = NULL;
DataAdaptor *pCanbusDataAdaptor = NULL;

/* Create instance and register it with the session bus
only if the service isn't already available. */
QDBusConnection connection = QDBusConnection::sessionBus();
if (!connection.interface()->isServiceRegistered(CANBUS_SERVICE)) {
    pCanbusData = new CanbusData(&a);
    pCanbusDataAdaptor = new DataAdaptor(pCanbusData);

    if (!connection.registerService(CANBUS_SERVICE)) {
        qFatal("Could not register service!");
    }

    if (!connection.registerObject(CANBUS_PATH, pCanbusData)) {
        qFatal("Could not register CanbusData object!");
    }
}

com::canbus::Data dataInterface(CANBUS_SERVICE
                                ,CANBUS_PATH,connection);

DbusHandler dbusHandler(dataInterface, &a);
```

Esimerkkikoodi 2. Ohjelman main-funktio ja D-Bus-alustus.

6.1.2 Interface- ja Adaptor-koodin generointi

D-Bus-koodin generoimisessa käytetään automaattista koodingenerointityökalua. Palvelinohjelma ja asiakasrajapinta käyttävät samaa D-Bus-rajapintaa ja palveluita, joten on järkevää toteuttaa ja päivittää näiden koodia olemassa olevilla automaattisilla työkaluilla. Esimerkiksi kun ohjelmointityön aikana on tarvetta päivittää tai lisätä jokin signaali palvelinohjelmasta rajapintaan, tarvitsee se kirjoittaa vain yhteen paikkaan koodissa. Tämän jälkeen työkalut päivittävät sen automaattisesti myös asiakasrajapintaan ja koko ohjelmistoprojektin D-Bus-koodi pysyy ajantasalla. CanbusData-luokkaan on toteutettu kaikki funktiot, slot-funktiot, signaalit ja ominaisuudet, jotka halutaan toteuttaa D-Bus-rajapintaan. Tästä luokasta generoidaan QAbstractAdaptor- ja QAbstractInterface-luokkaan pohjautuvat luokat. Qt sisältää ohjelmat qdbuscpp2xml ja qdbusxml2cpp, joilla voi generoida C++-koodia tai XML-tiedoston. Ohjelma qdbuscpp2xml lukee C++-koodi- tai otsikkotiedoston joka sisältää QObject-luokasta perittyä koodia ja parsii tästä XML-tiedoston. Esimerkkikoodissa 3 on ote tässä työssä generoidusta XML-tiedostosta. Se sisältää D-Bus-rajapinnan osoitteen, metodit ja signaalit sekä näiden datatyypit.

```
<!DOCTYPE node PUBLIC "-//freedesktop//DTD D-BUS Object Introspection
1.0//EN"
"http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">
<node>
  <interface name="com.canbus.Data">
    <property name="users" type="as" access="read"/>
    <signal name="userAdded">
      <arg name="user" type="s" direction="out"/>
    </signal>
    <method name="sendFJB1_AI_1">
      <arg name="data" type="u" direction="in"/>
    </method>
</interface>
</node>
```

Esimerkkikoodi 3. CanbusData-luokasta generoitu XML-tiedosto.

Ohjelma qdbusxml2cpp lukee XML-tiedoston ja parsii tästä Interface- ja Adaptor C++ -koodia. Esimerkkikoodissa 4 on ote generoidusta adaptor-luokasta.

```
/*** header ***/
class DataAdaptor: public QDBusAbstractAdaptor
{
    Q_OBJECT
    Q_CLASSINFO("D-Bus Interface", "com.canbus.Data")
    Q_CLASSINFO("D-Bus Introspection", ""
"  <interface name=\"com.canbus.Data\">\n"
"    <property access=\"read\" type=\"as\" name=\"users\"/>\n"

public: // PROPERTIES
    Q_PROPERTY(QStringList users READ users)
    QStringList users() const;


/*** source ***/
QStringList DataAdaptor::users() const
{
    // get the value of property users
    return qvariant_cast< QStringList >(parent()->property("users"));
}
```

Esimerkkikoodi 4. Generoitu DataAdaptor-luokka.

Qt:n qmake-ohjelma kutsuu qdbusxml2cpp- ja qdbuscpp2xml-ohjelmia, ja nämä parsivat D-Bus-koodin. Projektin pro-tiedostoon on lisätty generointityökalujen kutsut. Esimerkkikoodissa 5 määritellään ensiksi kansiot, joihin generoitu koodi kopioidaan. Tämän jälkeen system-kutsulle annetaan parametriksi generointiohjelman nimi ja sen parametrit. System-kutsu kutsuu qmake-esikäännöksen aikana sille parametriksi annettuja ohjelmia. Lopuksi generoitu koodi kopioidaan myös asiakasrajapinta-projektin kansioon Linuxin cp-ohjelmalla.

```
# the path where the dbus interface and adaptor code will be generated
DBUS_TARGET_PATH = canopendbus_generated
CLIENT_PATH = ../client_library/
CLIENT_TARGET_PATH = ../client_library/$$DBUS_TARGET_PATH
USER = $$system(whoami)

# create a folder if it does not exist
system(mkdir -p $$DBUS_TARGET_PATH)

# create xml file from CanbusData.h
system(qdbuscpp2xml CanbusData.h -A -o
$$DBUS_TARGET_PATH/CanbusData.xml)

# create interface and adaptor classes from xml file
system(qdbusxml2cpp $$DBUS_TARGET_PATH/CanbusData.xml -a
$$DBUS_TARGET_PATH/CanbusDataAdaptor)
system(qdbusxml2cpp $$DBUS_TARGET_PATH/CanbusData.xml -p
$$DBUS_TARGET_PATH/CanbusDataInterface)

# create dbus path to client library if it does not exist
system(mkdir -p $$CLIENT_TARGET_PATH)
system(cp $$DBUS_TARGET_PATH/* $$CLIENT_TARGET_PATH)
```

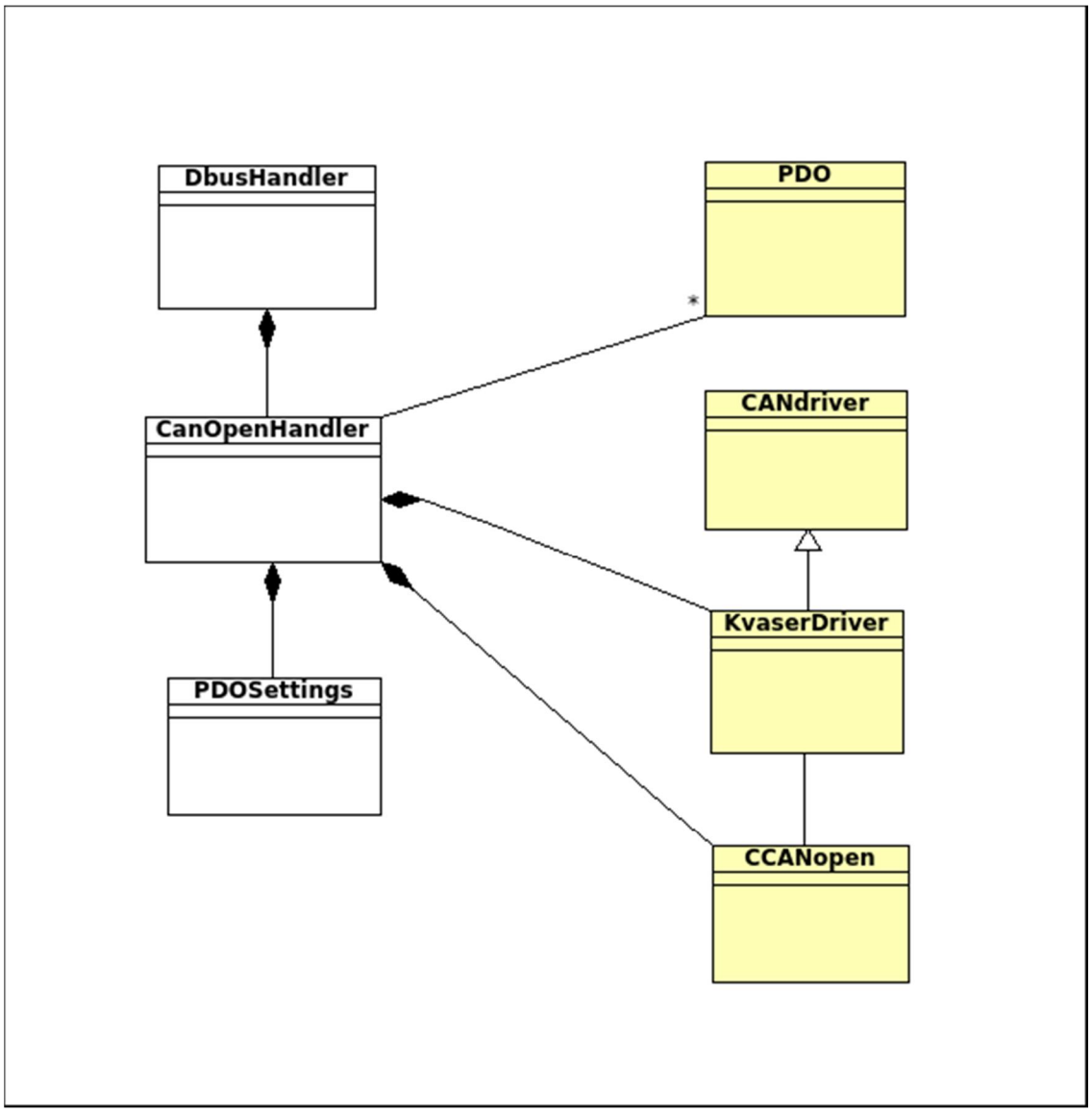
Esimerkkikoodi 5. D-Bus koodin generointi pro-tiedoston määrittelyiden avulla.

6.1.3 CANopen-tiedonsiirto

CANopen-tiedonsiirrossa käytetään Metropolian koneautomaatiolaboration kehittämää Minicanopen-kirjastoa. CanOpenHandler-luokka sisältää kaiken CANopen-toiminnallisuuteen liittyvän koodin. Se on ainut luokka, joka on yhteydessä Minicanopen-kirjastoon. CanOpenHandler-luokka muodostaa CCANopen-luokasta olion ja aiemmin mainitut PDO-oliot, joiden osoitteet lisätään CCANopen-oliolle. CanOpenHandler muodostaa myös KvaserDriver-luokasta olion. KvaserDriver on abstraktista CANdriver-luokasta peritty luokka, joka toteuttaa sen funktiot. KvaserDriver on Kvaser CAN-laitteen ajurin rajapinta. Ohjelman kehitysvaiheessa käytettiin Kvaser USBCAN -laitetta. Myöhemmin kun ohjelma otetaan käyttöön

IVI-tietokoneessa, jossa on eri valmistajan CAN-laite, sille täytyy toteuttaa vastaava CANdriver-luokasta peritty luokka.

PDOSettings-luokka on lisätty kuvaan 7 havainnollistamaan sen yhteyttä CanOpen-Handler-luokkaan ja Minicanopen-kirjastoon. Luvussa 6.1.4 käsitellään enemmän PDOSettings-luokan käyttöä ja Minicanopen-kirjaston PDO-luokan käyttöä.



Kuva 7. UML-luokkakaavio MiniCANopen-kirjaston käytöstä ohjelmassa.

6.1.4 PDO-olioiden määrittelyt

CAN-väylältä luettavan ja kirjoitettavan datan asetukset on määritelty CANopen PDO -protokollaan. PDO:ssa määritellään COBID ja sen pituus tavuina sekä annetaan muuttujan osoite, johon tietty PDO:ssa muuttuva data kirjoitetaan. Nämä tiedot sijaitsevat tpdo.csv- ja rpdo.csv-tiedostoissa, joista ne luetaan ohjelman käynnistyessä. Liitteessä 3 ja 4 on TPDO- ja RPDO-viestien tiedot.

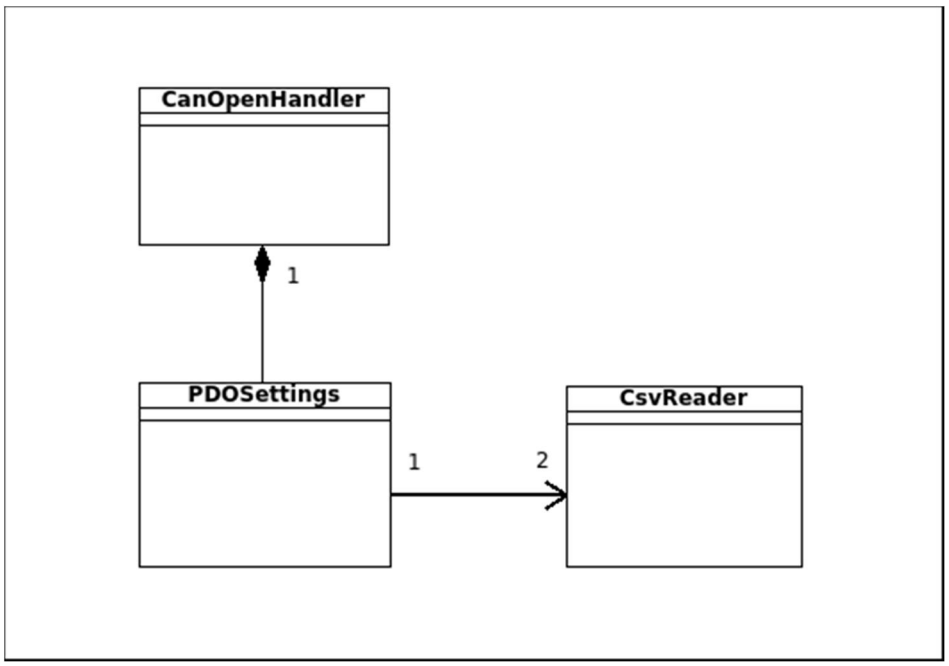
Tiedoston lukemista varten on toteutettu CsvReader-luokka. CsvReader-luokka lukee tiedoston rivi kerrallaan ja parsii siitä kaksiulotteisen merkkijonotaulukon. CsvReader-oliota muodostaessa annetaan sen rakentajaan parametreina tiedoston sijainti, erotinmerkki, joka on tässä tapauksessa puolipiste, rivi, mistä aloitetaan parsinta, sekä osoite PDOSettings-luokassa muodostettuun QVector<QStringList>-tyyppiseen olioon. CsvReader-luokan rakentajakutsun jälkeen täytyy sen jäsenfunktiota startParse vielä kutsua. Tämä funktio aloittaa tiedoston parsinnan. Esimerkkikoodissa 6 on havainnollistettu CsvReader-luokan käyttöä.

```
CsvReader csvReader(sg.value("canopen/tpdo").toString()
                        ,";"
                        ,1
                        ,&strTPDOList);

csvReader.startParse();
```

Esimerkkikoodi 6. CsvReader-luokan käytöstä esimerkki.

Kuvassa 8 on havainnollistettu PDO-asetusten lukemiseen yhteydessä olevia luokkia. CsvReader-olioita muodostetaan kaksi kappaletta, yksi TPDO-asetuksille ja yksi RPDO-asetuksille.



Kuva 8. PDOSettings- ja CsvReader-luokat ja näiden välinen yhteys.

PDOSettings-luokka sisältää kaikki PDO-protokollaan liittyvät tiedot. CanOpenHandler-luokasta käytetään näitä tietoja PDOSettings-luokan julkisten jäsenfunktioiden avulla. CanOpenHandler-luokassa muodostetaan dynaaminen taulukko, jossa pidetään PDO-olioiden osoitteita muistissa. Silmukassa jokaiselle PDO-oliolle varataan muistia sekä alustetaan sen asetukset. Esimerkkikoodissa 7 on ote koodista, jossa alustetaan TPDO-oliot for-silmukassa. RPDO-olioiden alustus on toteutettu samalla tavalla, mutta omassa funktiossaan. Kun CANopen-alustukset on tehty, alustetaan CAN-ajuri ja annetaan sen osoite CCANopen-oliolle. Esimerkkikoodissa 8 on havainnollistettu CAN-ajurin alustusta.

```
for(unsigned int i=0; i<pdoSettings_->getTPDOsize(); i++) {
    // 1. alloc TPDOs
    TPDO_[i] = new PDO(pdoSettings_->getTPDOcobid(i));
    // 2. set length every PDOs
    TPDO_[i]->set_len(8);
    // 3. alloc readed data list elements
    QVector<bool> tmpB;
    tpdoMappedBits_.append(tmpB);
    QVector<unsigned short> tmpU;
    tpdoMappedUshortBytes_.append(tmpU);
    QVector<unsigned char> tmpA;
    tpdoMappedUcharBytes_.append(tmpA);

    // 4. map variables to PDO
    for(unsigned int j=0; j<pdoSettings_->getTPDOindexSize(i); j++) {
        // 4.1 mapbitcopy(bit, PDO) <- bit variables
```

```
        if(pdoSettings_->getTPDOmsgType(i,j) == "bool") {
            tpdoMappedBits_[i].append(false);
            TPDO_[i]->mapbitcopy(pdoSettings_->getTPDOMsgbitIndex(i,j)
, tpdoMappedBits_[i][j]);
        }
        // 4.2 mapcopy(byte, PDO) <- byte variables
        else if(pdoSettings_->getTPDOmsgType(i,j) == "unsigned char")
{
            tpdoMappedUcharBytes_[i].append(0);
            TPDO_[i]->mapcopy(pdoSettings_->getTPDOMsgByteIndex(i,j)
, tpdoMappedUcharBytes_[i][j]);
        }
        // 4.3 mapcopy(byte, PDO) <- 2 byte variables
        else if(pdoSettings_->getTPDOmsgType(i,j) == "unsigned short")
{
            tpdoMappedUshortBytes_[i].append(0);
            TPDO_[i]->mapcopy(pdoSettings_->getTPDOMsgByteIndex(i,j)
, tpdoMappedUshortBytes_[i][j]);
        }
    }
    // 5. add TPDOs to the pdolist
    can_.add_tpdo(*TPDO_[i]);
}
```

Esimerkkikoodi 7. PDO-olioiden muistinvaraus ja alustus.

```
if (driver_.open(settings_.value("canbus/speed_bps").toUInt(0),
                 settings_.value("canbus/mode").toUInt(0),
                 settings_.value("canbus/port").toUInt(0),
                 settings_.value("canbus/silent").toBool()) == 0) {
    // Reset driver after error, rx debug, tx debug
    driver_.set_debug(CANDRV_DEBUG_RESET);

    // Exit after critical error
    can_.set_debug(CAN_DEBUG_EXIT);
    can_.load_driver(driver_);
    isCANbusOpen = true;

    qDebug() << "void CanOpenHandler::initializeCanBus(): CAN IS
OPEN";

    return true;
}
else
    return false;
```

Esimerkkikoodi 8. CAN-laiteajurin alustus ja ajurin lataaminen CCANopen-oliolle.

6.1.5 Ohjelman ajastimien alustus ja käynnistys

Ohjelmassa käytetään neljää eri ajastinta ja niiden käyttötarkoitukset ovat seuraavat:

- canReadTimer_, käytetään CANopen-pakettien lukemiseen. Ajastinaika 20 ms.
- heartBeatTimer_, käytetään CANopen Hearbeat -protokollan mukaisen viestin lähettämiseen. Ajastinaika 800 ms.
- canEventTimer_, aina kun tämä ajastin liipaisee, ohjelma lähettää TPDO-viestit väylälle, oli ne niiden data muuttunut tai ei. Ajastinaika 2 s.
- canInhibitTimer_, tätä käytetään mittaamaan aikaa, koska edellisen kerran lähetettiin TPDO-viesti. Jos tämä ajastin ei ole kerinnyt liipaisemaan ja käyttäjä haluaa lähettää TPDO-viestin, viestiä ei lähetetä.

Kun ohjelma on saanut alustettua sekä CAN-ajurin että CANopen-asetukset onnistuneesti, ajastimet käynnistetään ja ohjelma siirtyy normaaliin toimintatilaan. Jos alustukset eivät onnistuneet, ohjelma lähettää D-Busille tästä virheviestin, sulkee itsensä ja tulostaa vielä ohjelman ulostuloon virheviestin. Esimerkkikoodissa 9 on havainnollistettu kyseisiä ehtolauseita.

```
if(initializeCANopen()) {
    if(initializeCanBus()) {
        canReadTimer_->start(canReadTime_ms);
        heartbeatTimer_->start(heartBeatTime_ms);
        canEventTimer->start(canEventTime_ms);
        canInhibitTimer.start();
    }
    else {
        qFatal("Fail to initialize CANdriver");
    }
}
else {
    qFatal("Fail to Initialize CANopen");
}
```

Esimerkkikoodi 9. CANopen-ajastitimien alustus ja käynnistys.

6.1.6 Datan vastaanotto CAN-väylältä

CanOpenHandler-luokassa on slot-funktio readCanMessages, joka suoritetaan 20 ms välein. Tässä funktiossa luetaan CAN-laitteen puskuri tyhjäksi, käsitellään CANopen-viestit ja lopuksi lähetetään PDO-viestit D-Busille. Funktiossa iteroidaan kaikki CAN-

viestit ja jokainen viesti lähetetään Qt:n signaalina eteenpäin. Signaalissa on parametrina data, joka on QVariant-tyyppiä sekä datan indeksinumero. Esimerkkikoodissa 10 on havainnollistettu lähetettävän signaalin esittelyä.

```
void notifyNewMessage(QVariant data, unsigned int index);
```

Esimerkkikoodi 10. CanOpenHandler-luokan CAN-datan lähetys-signaali-funktio.

DbusHandler-luokassa on slot-funktio receiveFromCan, jossa CanOpenHandler-luokasta lähetetetty signaali vastaanotetaan ja käsitellään. Käsittelyssä on switch-case-rakenne, jossa tulevan viestin indeksinumeroa vertaillaan valmiiksi määriteltyihin enumeraatioihin. Tämän jälkeen viesti lähetetään D-Busille. Esimerkkikoodissa 11 on havainnollistettu vastaanotettavan datan switch-case-rakennetta.

```
switch (index) {
case CityCar::FJB1_DO_0:
    dataInterface_.sendFJB1_DO_0(data.toBool());
    break;
case CityCar::FJB1_DO_1:
    dataInterface_.sendFJB1_DO_1(data.toBool());
    break;
```

Esimerkkikoodi 11. Switch-case-rakenne CAN-viestien välittämisestä eteenpäin.

6.1.7 Datan vastaanotto D-Busilta

D-Busilta tulevat viestit, jotka on tarkoitus lähettää CAN-väylälle, vastaanotetaan ensiksi DbusHandler-luokassa omiin privaatti slot-funktioihin. Näistä funktioista viestit ohjataan eteenpäin CanOpenHandler-luokalle, jossa funktiossa handleAndSendAppMessages reititetään viestit enumeraatioiden perusteella. Käsittelyssä viesteillä on kullakin oma indeksinumeronsa, joiden mukaan switch-case-rakenteella niiden data päivitetään lähetettäviin PDO-viesteihin. Funktion handleAndSendAppMessages lopussa kaikki PDO:t lähetetään CAN-väylälle. Esimerkkikoodissa 12 on havainnollistettu D-Busilta tulevien viestien käsittelyä.

```
switch (index) {
case CityCar::FrontGlassHeaterOn:
    handleOutputData(data, index);
    break;
case CityCar::RearGlassHeaterOn:
    handleOutputData(data, index);
    break;
```

Esimerkkikoodi 12. D-Busilta tulevien viestien reititys ja käsittely.

6.1.8 Ethernet-verkon tiedonsiirto UDP-protokollalla

Työn aikana päätettiin mittariston ja apukuljettajan tietokoneen tiedonsiirto toteuttaa Ethernet-verkon avulla. Tähän ratkaisuun päädyttiin sen takia, että IVI-tietokoneen täytyisi muutenkin reitittää CAN-väylältä tuleva data mittariston ja apukuljettajan tietokoneisiin. IVI-tietokoneen täytyisi siis reitittää CAN-väylältä tulevat viestit Ethernet-verkkoon. Tiedonsiirtoprotokollaksi valittiin TCP/IP-prokollaperheestä tuttu UDP-protokolla. UDP-protokolla on yksinkertainen, epäluotettava, kevyt ja yhteydetön tiedonsiirtoprotokolla (19).

UDP-protokollan toteutuksessa käytettiin Qt:n QUdpSocket-luokkaa, josta muodostettiin kantaluokka UdpNetwork. UdpNetwork-luokka toteuttaa UDP-yhteyden alustuksen, datan lähetyksen ja vastaanoton. UdpNetwork-luokasta muodostetaan aliluokka UdpCanData, jossa toteutetaan CAN-datan lähetys Ethernet-verkkoon DbusHandler-luokan avulla. UdpNetwork-kantaluokka toteutettiin siitä syystä, että IVI-järjestelmän tiedonsiirtoa tullaan vielä jatkokehittämään ja tätä samaa luokkaa tullaan käyttämään kantaluokkana useille luokille tulevaisuudessa.

UdpCanData-luokasta muodostetaan olio DbusHandler-luokan rakentajassa. Oliolle annetaan parametrina QObject-isäntäolion osoite, UDP-portti sekä IP-osoite mihin viestit lähetetään. Viestejä lähetetään sendPacket-funktiolla, johon annetaan parametriksi indeksinumero ja data. Esimerkkikoodissa 13 on havainnollistettu UdpCanData-luokan sendPacket-funktion määrittelyä.

```
void sendPacket(CityCar::CanData dataType, QVariant data);
```

Esimerkkikoodi 13. UdpCanData-luokan sendPacket-funktion määrittely.

6.1.9 Ohjelman asetukset, Qsettings

Ohjelman asetukset luetaan sen käynnistyessä conf-tiedostosta, jonka polku on /home/user/.config/CCCP/canopendbus.conf Ubuntu Linux -käyttöjärjestelmissä. Qt sisältää luokan QSettings, jolla voi lukea ja kirjoittaa asetuksia ohjelmasta. Asetustiedostoon on tallennettu CAN-väylään liittyviä, CANopen-protokolliin liittyviä ja ohjelman yleisiä asetuksia. Esimerkiksi tpdo.csv- ja rpdo.csv-tiedostojen polut on tallennettu kyseiseen asetustiedostoon. Esimerkkikoodissa 14 on havainnollistettu asetusten lukemista QSettings-tyyppisen olion avulla.

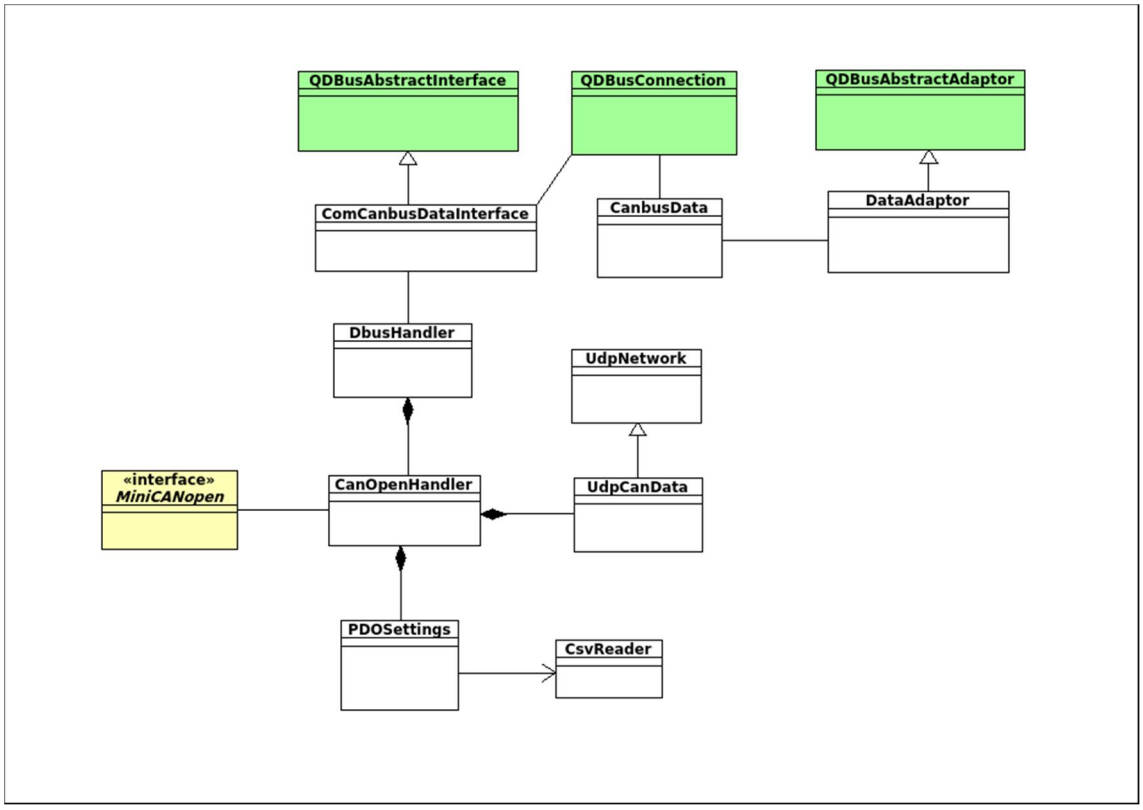
```
QSettings settings_("CCCP", "canopendbus");
bool okMyNodeID = false;
myNodeID = settings_.value("canopen/nodeid").toUInt(&okMyNodeID);
```

Esimerkkikoodi 14. Asetuksien lukeminen tiedostosta QSettings-olion avulla.

QSettings-oliolle varataan muistia pinosta aina, kun sitä tarvitaan. Sen rakentajaan annetaan parametrina organisaation nimi, joka on tässä tapauksessa CCCP, ja toisena parametrina ohjelman nimi. Näillä tiedoilla QSettings osaa automaattisesti hakea käyttöjärjestelmän ohjelmien asetuskansiosta oikean tiedoston. Arvo luetaan value-funktiolla, joka palauttaa merkkijonon ja johon annetaan parametriksi avain merkkijonona.

6.1.10 Ohjelman UML-kaavio

Kuvassa 9 näkyy palvelinohjelman luokkakaavio. Vihreällä taustalla olevat luokat ovat QtDBus-moduulin luokkia. Minicanopen-rajapinta on keltaisella taustalla.



Kuva 9. Palvelin ohjelman UML-luokkakaavio.

6.2 Asiakasohjelman rajapinta

Ohjelmointirajapinnan käytön tulisi olla selkeä ja helppokäyttöinen. Palvelinohjelma lähettää signaalin kun CAN-väylän datassa tapahtuu muutoksia. Esimerkiksi kun RJB:n analogiselta sisääntulolta tulee uusi tieto, palvelinohjelma lähettää signaalin D-Busin läpi, ja ohjelmissa, joissa on toteutettu C++:ssa slot-funktio tai QML:ssa Connection Item, saavat välittömästi tämän uuden tiedon.

Toiminnallisuuksia mitä rajapinnalta vaaditaan:

- Rajapintaa tulee käyttämään useampi ohjelma samaan aikaan.
- Rajapinnan tulee olla selkeä ja helppokäyttöinen.

- Rajapintaa täytyy pystyä käyttämään sekä C++ että QML-kielellä.
- Rajapinta toteutetaan Qt:n signal- ja slot-mekanismilla, asiakasohjelma yhdistää rajapinnan signaalin omaan slot-funktioonsa tai käsittelee muulla tavalla signaalin.
- Rajapinnan signaalien nimeämisessä käytetään samaa nimeämistä kuin muussa kaupunkiauton korisähköjärjestelmän signaalien ja toimilaitteiden nimeämisessä.
- Kun asiakasohjelma käyttää rajapintaa, niin rajapinnassa muodostetaan automaattisesti D-Bus-yhteys. D-Busin käyttö ei näy käyttäjälle mitenkään.

6.2.1 Linuxin aliohjelmakirjastot

Linuxissa on kahdentyyppisiä kirjastoja, ei-jaettuja kirjastoja ja jaettuja kirjastoja. Ei-jaetut kirjastot ovat lähdekoodista käännettyjä objektikooditiedostoja. Näiden tiedostojen pääte on ”.a”, joka tulee archive-sanasta. Kun käytetään ei-jaettuja kirjastoja, kääntäjä kopioi kirjastoissa olevan koodin käännettävään ohjelmaan mukaan. Ei-jaettujen kirjastojen käyttö kasvattaa käännetyn ohjelman kokoa. Näin myös muistin käyttö kasvaa, kun kaikille ohjelmille kopioidaan sama koodi ei-jaetusta kirjastosta. (1, s. 37.)

Jaettu kirjasto on objektikoodia käyttöjärjestelmässä, joka on käytössä kaikille tai ainakin useimmille ohjelmille. Jaetun kirjaston tarkoituksena on vähentää ohjelman koodin määrää sekä pienentää käännettävän ohjelman kokoa ja muistin käyttöä. Linuxissa jaetun kirjaston pääte on ”.so”, joka tulee sanoista shared object. Siinä missä ei-jaetun kirjaston koodi kopioidaan jokaiseen ohjelmaan, jaetun kirjaston koodia ei kopioida jokaiseen ohjelmaan. Kun ohjelma käyttää jaettua kirjastoa, kirjasto ladataan muistiin kaikkien ohjelmien käytettäväksi. Tällä tavalla käännettävän ohjelman koko sekä muistinkäyttö on pienempi. (1, s. 38–40.) Tässä työssä asiakasohjelman rajapinnasta käännetään jaettu kirjasto.

Qt Creator:ssa voi muodostaa graafisella työkalulla C++-kirjastoprojektin, joka tekee tarvittavat määrittelyt jaetun kirjaston muodostamiseksi. Qt Creator muodostaa global.h-otsikkotiedoston jossa on määrittelyt kirjaston näkyvyyksille. Kun Qt:ssa halutaan asettaa luokkia tai funktioita näkymään esimerkkikoodissa 15 muodostetulla näkyvyysmääreellä, ne määritellään esimerkkikoodin 16 mukaisella tavalla.

```
#include <QtCore/qglobal.h>
#if defined(CANOPENDBUS_LIBRARY)
#  define CANOPENDBUSSHARED_EXPORT Q_DECL_EXPORT
#else
#  define CANOPENDBUSSHARED_EXPORT Q_DECL_IMPORT
#endif
```

Esimerkkikoodi 15. Qt Creator:n automaattisesti muodostama otsikkotiedosto.

```
#include "CanOpenDbus_global.h"

class CANOPENDBUSSHARED_EXPORT CanOpenDbus : public QObject
```

Esimerkkikoodi 16. CanOpenDbus-luokan määrittely jaetuksi kirjastoksi.

6.2.2 Rajapinnan palvelut

Rajapinnan toteuttamisessa pystyttiin käyttämään samaa D-Bus-koodia kuin palvelinohjelmassa. Rajapintaan täytyi toteuttaa D-Bus-alustukset ja yhdistäminen palvelinohjelman D-Bus-palveluun, sekä D-Busilta tuleville signaaleille slot-funktiot ja näille Q_PROPERTY-määritykset, jotta niitä voidaan käyttää myös QML-koodissa. D-Bus-alustukset toteutettiin samalla tavalla kuin palvelinohjelmassa. Alla olevassa esimerkkikoodissa on Q_PROPERTY-makrojen määritys D-Busilta saapuville signaaleille. Liitteessä 2 on luettelo kaikista Q_PROPERTY-määrittelyistä rajapinnassa. Funktiot, joita halutaan käyttää rajapinnasta QML-kielellä, täytyy määritellä Q_INVOKABLE-makrolla. Makrot Q_PROPERTY ja Q_INVOKABLE ovat Qt:n metaoliojärjestelmän makroja. Näiden makrojen avulla Qt:n moc (Meta-Object Compiler) generoi automaattisesti C++-koodia, joka käännetään vasta tämän jälkeen käyttöjärjestelmän omalla kääntäjällä. Esimerkkikoodissa 17 on havainnollistettu rajapinnan otsikkotiedoston määrittelyä.

```
// Property declaration
Q_PROPERTY(bool data READ fjb1_di_0 NOTIFY fjb1_di_0Changed)

// Get function
bool fjb1_di_0() { return m_FJB1_DI_0; }

// Receive slot for incoming D-Bus signal
void receiveFJB1_DI_0(const bool &data)
{
    m_FJB1_DI_0 = data;
```

```
    emit fjb1_di_0Changed();
}

// Q_INVOKABLE function declare
Q_INVOKABLE void setFrontGlassHeaterOn(const bool &data);
```

Esimerkkikoodi 17. Koodiesimerkki CanOpenDbus-luokan määrittelyistä.

Rajapintaan on toteutettu myös julkisia funktioita, joita kutsutaan kun käyttöliittymästä halutaan lähettää CAN-väylälle jotain tietoa. Nämä funktiot on listattu liitteessä 1. Funktioille annetaan parametrina haluttu muuttuja. Esimerkkikoodissa 18 on havainnollistettu rajapinnan julkisen funktion toteutusta.

```
void CanOpenDbus::setFrontGlassHeaterOn(bool data)
{
    dataInterface_->setFrontGlassHeaterOn(data);
}
```

Esimerkkikoodi 18. Rajapinnan funktion toteutus, jolla voidaan lähettää pyyntö etulasinlämmittimen päällekytkemiseksi.

6.2.3 Rajapintaesimerkki

Rajapintaa voidaan käyttää sekä C++-, että QML-kielellä. Kun halutaan lukea dataa rajapinnasta C++-kielellä, muodostetaan ensiksi CanOpenDbus-tyyppinen olio ja tämän jälkeen yhdistetään halutut signaalit omiin slot-funktioihin. Kun C++-kielellä halutaan ohjata jotain auton toimintoa, voidaan yksinkertaisesti vaan kutsua kyseistä rajapinnan funkiota. Esimerkkikoodissa 19 on havainnollistettu rajapinnan käyttöä C++-kielellä.

```
CanOpenDbus canOpenDbus(app);
canOpenDbus.setACfanSpeed(10);
```

Esimerkkikoodi 19. C++-koodiesimerkki rajapinnan funktion käytöstä.

Käytettäessä rajapintaa QML-koodista, täytyy ennen QML-koodin lataamista muodostaa CanOpenDbus-tyyppinen olio ja asettaa tämä olio käytettäväksi myös QML-koodin

puolelta. Tämä tapahtuu esimerkkikoodin 20 mukaisesti. QmlApplicationViewer-tyyppisen viewer-olion rootContext-tyypille asetetaan ominaisuudeksi nimi canOpenDbus sekä ylempänä muodostetun canOpenDbus-olion osoite.

```
CanOpenDbus canOpenDbus;
viewer.rootContext()->setContextProperty("canOpenDbus"
                                         ,&canOpenDbus);
```

Esimerkkikoodi 20. Rajapinnan käyttö QML-koodissa.

Esimerkkikoodissa 21 on havainnollistettu rajapinnan käyttöä QML-kielellä. Koodissa määritellään target-ominaisuudeksi esimerkkikoodissa 20 muodostettu canOpenDbus-olio. Signaali onFjb1_di_0Changed määrittelyn sisällä oleva koodi ajetaan, kun kyseisen signaalin tila vaihtuu.

```
Connections {
    // Target name was defined in the main.cpp
    target: canOpenDbus

    // This signal will emitted when the FJB1_DI_0 value changed
    onFjb1_di_0Changed: {

        // Print the FJB1_DI_0 value to console output
        console.log("FJB1_DI_0 changed: ", canOpenDbus.fjb1_di_0())
    }
}
```

Esimerkkikoodi 21. QML-koodiesimerkki datan lukemisesta rajapinnasta.

Esimerkkikoodissa 22 on havainnollistettu rajapinnan funktioiden käyttöä. Koodissa on määritelty Slider-tyyppinen komponentti. Kun Slider-komponentin onValueChanged-arvo muuttuu, ohjelma suorittaa rajapinnan setACtemp-funktion parametrin arvolla mikä riippuu sen hetkisestä Slider-komponentin asemasta.

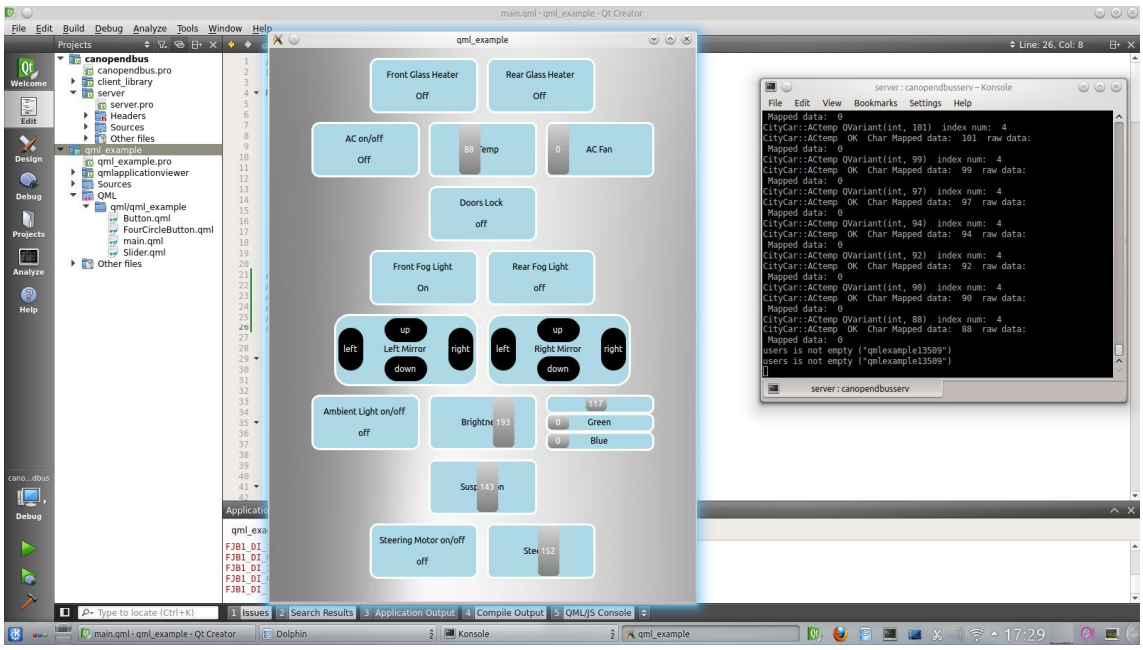
```
Slider {
    id: acTempSlider
    width: buttonWidth
    height: buttonHeight
    labelText: "AC Temp"
    minimum: 0
    maximum: 255
    onValueChanged: canOpenDbus.setACtemp(value)
}
```

Esimerkkikoodi 22. QML-koodiesimerkki rajapinnan funktion käytöstä.

6.3 Palvelinohjelman ja rajapinnan testaus

Testausta suoritettiin koko ohjelman kehityksen ajan aina jonkin ohjelmanosan valmistuttua. Lopuksi vielä testattiin koko ohjelman ja rajapinnan toimintaa testiohjelmilla, joita tullaan käyttämään valmiissa IVIssä.

Testausta varten ohjelmoitiin kaksi testiohjelma, jotka loppujen lopuksi ovat myös esimerkkiohjelmia jatkoa varten. Testiohjelmien koodi sijaitsee projektin canopendbus/src/examples-kansiossa. Testiohjelmilla testattiin kirjoitusta CAN-väylälle D-Busin välityksellä sekä CAN-väylältä lukemista D-Busin välityksellä. Lukutestausohjelman käyttöliittymä toteutettiin QML-kielellä, johon tehtiin lista, mihin CAN-väylän data päivittyy automaattisesti. Tätä ohjelmaa testattiin oikeassa ympäristössä, jossa oli kaupunkiauton sähköjärjestelmän ohjainlaitteita kytkettyinä. Kaikkia CAN-väylän tietoja ei päästy testaamaan oikeassa ympäristössä, koska kaikkia laitteita ei ole vielä kytketty. Kuvassa 10 on kuvaruutukaappaus ohjelman testauksesta.



Kuva 10. Testauksesta kuva.

Kirjoitustestauksen käyttöliittymä toteutettiin myös QML-kielellä, ja siinä on napit kaikelle tiedolle, mitä D-Busin välityksellä on tarkoitus lähettää CAN-väylälle. Tätä ohjelmaa testattiin oikeassa ympäristössä. Auton korielektroniikan ohjainlaite EPEC 2024 vastaanotti QML-testiohjelman lähettämiä viestejä.

Kummassakaan testausvaiheessa ei vältytty virheiltä niinpä testauksen välissä korjattiin virheitä ohjelmasta ja palattiin taas takaisin testaamaan, kunnes kaikki toimi halutulla tavalla.

7 Yhteenveto

Työssä oli tavoitteena luoda ohjelma, joka välittää dataa CAN-väylältä käyttöjärjestelmän muille ohjelmille D-Bus-prosessien välisen tiedonsiirtomekanismin avulla sekä ohjelmointirajapinta tälle ohjelmalle. Työ toteutettiin käyttämällä QtDBus-moduulia. Työ saatiin valmiiksi ja työn tuloksena syntyi ohjelma sekä rajapinta, jolla ohjelmaa voidaan käyttää. Työn alkuvaihe käytettiin D-Bus- ja CANopen-protokollaperheen teorian opiskeluun. Tämän jälkeen ohjelma toteutettiin. Ongelmat, jotka tulivat eteen työtä toteutettaessa, saatiin ratkaistua.

Ohjelmointityön aikana ja tuloksia tarkastellessa tuli esiin muutamia kehityskohteita tai asioita, joita olisi voinut tehdä toisin. Yksi kehityskohde olisi siirtyminen Minicanopen-kirjastosta CanFestival-kirjaston käyttöön. Tämä siitä syystä, että Minicanopen-kirjasto on osittain keskeneräinen ja buginen eikä sitä ei ole kehitetty paljoakaan sen jälkeen, kun se 2000-luvun alussa luotiin. CanFestival-kirjaston ympärillä taas on aktiivisia kehittäjiä, ja se on käytännössä valmis. Se on myös yleisesti käytössä käytännön projekteissa. CanFestival-kirjastoon löytyy myös valmiita graafisia työkaluja objektikirjaston parametrien asetukseen, kun taas Minicanopen-kirjastossa ei ole edes valmista objektikirjastoa. CanFestival-kirjastoon löytyy myös enemmän valmiita laiteprofiileita eri valmistajien CAN-laitteille.

Toinen kehityskohde olisi tehdä ohjelmasta geneerisempi. Tällä hetkellä D-Busin läpi siirretään pelkästään kaupunkiauton sähköjärjestelmään liittyvää dataa. Kun geneerisemmässä ohjelmassa D-Busin läpi voitaisiin siirtää esimerkiksi suoraan CANopen-dataa. D-Busin läpi voitaisiin siirtää suoraan PDO-protokollaviestejä ja ne voitaisiin vasta asiakasrajapinnassa osoittaa oikeaan datan kuluttujaan. Kolmas kehityskohde voisi olla ohjelman lisääminen jonkin Linux-jakelun pakettivarastoon, jolloin ohjelman päivitys olisi joustavampaa ja loogisempaa.

Lähteet

1 Intel In-Vehicle Infotainment. Verkkodokumentti. Intel. <http://www.intel.com/content/www/us/en/intelligent-systems/in-vehicle-infotainment/in-vehicle-infotainment-in-car-entertainment-with-intel-inside.html>. Luettu 3.2.2013.

2 Tesla Model S features & specs. Verkkodokumentti. Tesla Motors. <http://www.teslamotors.com/models/features#/interior>. Luettu 3.2.2013.

3 Mitchell, Mark, Oldham, Jeffrey & Samuel, Alex. 2001. Advanced Linux Programming. USA: New Riders.

4 Pennington, Havoc, Wheeler, David, Palmieri, John & Walters, Colin. D-Bus Tutorial. Verkkodokumentti. <http://dbus.freedesktop.org/doc/dbus-tutorial.html>. Luettu 3.2.2013.

5 Introduction to D-Bus. 2011. Verkkodokumentti. freedesktop.org. <http://www.freedesktop.org/wiki/IntroductionToDBus>. 16.12.2011. Luettu 3.2.2013.

6 Pennington, Havoc, Carlsson, Anders, Larsson, Alexander, Herzberg, Sven, McVittie, Simon & Zeuthen, David. 2013. D-Bus Specification. Verkkodokumentti. <http://dbus.freedesktop.org/doc/dbus-specification.html>. 22.2.2013. Luettu 3.2.2013.

7 Using D-Bus. Telepathy Developer's Manual. Verkkodokumentti. freedesktop.org. <http://telepathy.freedesktop.org/doc/book/sect.basics.dbus.html>. Luettu 3.2.2013.

8 QtDBus module dokumentaatio. Qt 4.8 dokumentaatio. Verkkodokumentti. Qt Project. <http://qt-project.org/doc/qt-4.8/qtdbus.html>. Luettu 3.2.2013.

9 QDBusConnection luokan dokumentaatio. Qt 4.8 dokumentaatio. Verkkodokumentti. Qt Project. <http://qt-project.org/doc/qt-4.8/qdbusconnection.html>. Luettu 3.2.2013.

10 QDBusAbstractAdaptor luokan dokumentaatio. Qt 4.8 dokumentaatio. Verkkodokumentti. Qt Project. <http://qt-project.org/doc/qt-4.8/qdbusabstractadaptor.html>. Luettu 15.4.2013.

11 QDBusAbstractInterface luokan dokumentaatio. Qt 4.8 dokumentaatio. Verkkodokumentti. Qt Project. <http://qt-project.org/doc/qt-4.8/qdbusabstractinterface.html>. Luettu 15.4.2013.

12 Saha, Heikki. 2006. CANopen perusteet. Verkkodokumentti. <http://www.canopen.fi/artikkelit/CANopen.pdf>. Luettu 27.2.2013.

13 Higher Layer Protocols. 2013. Verkkodokumentti. Kvaser AB. <http://www.kvaser.com/en/about-can/higher-layer-protocols.html>. Luettu 27.2.2013.

14 Application Domains. 2013. Verkkodokumentti. CAN in Automation. <http://www.can-cia.org/index.php?id=30>. Luettu 27.2.2013.

15 Pfeiffer, Olaf, Ayre, Andrew & Keydel, Christian. 2003. Embedded Networking with CAN and CANopen. USA: RTC Books.

16 CANopen device and application profiles. 2013. Verkkodokumentti. CAN in Automation. <http://www.can-cia.org/index.php?id=systemdesign-profiles>. Luettu 27.2.2013.

17 CanFestival. 2001. Verkkodokumentti. CanFestival-yhteisö. <http://www.canfestival.org/>. Luettu 3.2.2013.

18 The CanFestival CANopen stack manual. Verkkodokumentti. CanFestival-yhteisö. <http://dev.automforge.net/CanFestival-3/raw-file/884a60cbb83e/objdictgen/doc/manual_en.pdf>. Luettu 3.2.2013.

19 QUdpSocket luokan dokumentaatio. Qt 4.8 dokumentaatio. Verkkodokumentti. Qt Project. <http://qt-project.org/doc/qt-4.8/qudpsocket.html>. Luettu 25.4.2013.

Rajapinnan funktiot

Q_INVOKABLE void setFrontGlassHeaterOn(const bool &data);
Q_INVOKABLE void setRearGlassHeaterOn(const bool &data);
Q_INVOKABLE void setACon(const bool &data);
Q_INVOKABLE void setACtemp(const int &data);
Q_INVOKABLE void setACfanSpeed(const int &data);
Q_INVOKABLE void setDoorsLockOn(const bool &data);
Q_INVOKABLE void setDriveLightsOn(const bool &data);
Q_INVOKABLE void setParkLightsOn(const bool &data);
Q_INVOKABLE void setFrontFogLIghtsOn(const bool &data);
Q_INVOKABLE void setRearFogLightsOn(const bool &data);
Q_INVOKABLE void setLeftMirrorUpOn(const bool &data);
Q_INVOKABLE void setLeftMirrorDownOn(const bool &data);
Q_INVOKABLE void setLeftMirrorLeftOn(const bool &data);
Q_INVOKABLE void setLeftMirrorRightOn(const bool &data);
Q_INVOKABLE void setRightMirrorUpOn(const bool &data);
Q_INVOKABLE void setRightMirrorDownOn(const bool &data);
Q_INVOKABLE void setRightMirrorLeftOn(const bool &data);
Q_INVOKABLE void setRightMirrorRightOn(const bool &data);
Q_INVOKABLE void setAmbientLightOn(const bool &data);
Q_INVOKABLE void setAmbienLightBrightness(const int &data);
Q_INVOKABLE void setAmbienLightRGB(const int &r, const int &g, const int &b);
Q_INVOKABLE void setSuspensionLevel(const int &data);
Q_INVOKABLE void setSteeringMotorOn(const bool &data);
Q_INVOKABLE void setSteeringMotorSpeed(const int &data);

Rajapinnan ominaisuudet

Q_PROPERTY(bool data READ fjb1_di_0 NOTIFY fjb1_di_0Changed)
Q_PROPERTY(bool data READ fjb1_di_1 NOTIFY fjb1_di_1Changed)
Q_PROPERTY(bool data READ fjb1_di_2 NOTIFY fjb1_di_2Changed)
Q_PROPERTY(bool data READ fjb1_di_3 NOTIFY fjb1_di_3Changed)
Q_PROPERTY(bool data READ fjb1_do_0 NOTIFY fjb1_do_0Changed)
Q_PROPERTY(bool data READ fjb1_do_1 NOTIFY fjb1_do_1Changed)
Q_PROPERTY(bool data READ fjb1_do_2 NOTIFY fjb1_do_2Changed)
Q_PROPERTY(bool data READ fjb1_do_3 NOTIFY fjb1_do_3Changed)
Q_PROPERTY(bool data READ fjb1_do_4 NOTIFY fjb1_do_4Changed)
Q_PROPERTY(bool data READ fjb1_do_5 NOTIFY fjb1_do_5Changed)
Q_PROPERTY(bool data READ fjb1_do_6 NOTIFY fjb1_do_6Changed)
Q_PROPERTY(bool data READ fjb1_do_7 NOTIFY fjb1_do_7Changed)
Q_PROPERTY(bool data READ fjb1_do_8 NOTIFY fjb1_do_8Changed)
Q_PROPERTY(bool data READ fjb1_do_9 NOTIFY fjb1_do_9Changed)
Q_PROPERTY(bool data READ fjb1_do_10 NOTIFY fjb1_do_10Changed)
Q_PROPERTY(bool data READ fjb1_do_11 NOTIFY fjb1_do_11Changed)
Q_PROPERTY(bool data READ fjb1_dio_0 NOTIFY fjb1_dio_0Changed)
Q_PROPERTY(bool data READ fjb1_dio_1 NOTIFY fjb1_dio_1Changed)
Q_PROPERTY(bool data READ fjb1_dio_2 NOTIFY fjb1_dio_2Changed)
Q_PROPERTY(bool data READ fjb1_dio_3 NOTIFY fjb1_dio_3Changed)
Q_PROPERTY(bool data READ fjb1_dio_4 NOTIFY fjb1_dio_4Changed)
Q_PROPERTY(bool data READ fjb1_dio_5 NOTIFY fjb1_dio_5Changed)
Q_PROPERTY(bool data READ fjb1_dio_6 NOTIFY fjb1_dio_6Changed)
Q_PROPERTY(bool data READ fjb1_dio_7 NOTIFY fjb1_dio_7Changed)
Q_PROPERTY(unsigned int data READ fjb1_ai_0 NOTIFY fjb1_ai_0Changed)
Q_PROPERTY(unsigned int data READ fjb1_ai_1 NOTIFY fjb1_ai_1Changed)
Q_PROPERTY(unsigned int data READ fjb1_ai_2 NOTIFY fjb1_ai_2Changed)
Q_PROPERTY(unsigned int data READ fjb1_ai_3 NOTIFY fjb1_ai_3Changed)
Q_PROPERTY(unsigned int data READ fjb1_ai_4 NOTIFY fjb1_ai_4Changed)
Q_PROPERTY(unsigned int data READ fjb1_ai_5 NOTIFY fjb1_ai_5Changed)
Q_PROPERTY(unsigned int data READ fjb1_ai_6 NOTIFY fjb1_ai_6Changed)
Q_PROPERTY(unsigned int data READ fjb1_ai_7 NOTIFY fjb1_ai_7Changed)
Q_PROPERTY(bool data READ fjb2_do_0 NOTIFY fjb2_do_0Changed)
Q_PROPERTY(bool data READ fjb2_do_1 NOTIFY fjb2_do_1Changed)
Q_PROPERTY(bool data READ fjb2_do_2 NOTIFY fjb2_do_2Changed)
Q_PROPERTY(bool data READ fjb2_do_3 NOTIFY fjb2_do_3Changed)
Q_PROPERTY(bool data READ fjb2_dio_0 NOTIFY fjb2_dio_0Changed)
Q_PROPERTY(bool data READ fjb2_dio_1 NOTIFY fjb2_dio_1Changed)
Q_PROPERTY(bool data READ fjb2_dio_2 NOTIFY fjb2_dio_2Changed)
Q_PROPERTY(bool data READ fjb2_dio_3 NOTIFY fjb2_dio_3Changed)
Q_PROPERTY(bool data READ fjb2_di_0 NOTIFY fjb2_di_0Changed)

Q_PROPERTY(bool data READ fjb2_di_1 NOTIFY fjb2_di_1Changed)
Q_PROPERTY(bool data READ fjb2_di_2 NOTIFY fjb2_di_2Changed)
Q_PROPERTY(bool data READ fjb2_di_3 NOTIFY fjb2_di_3Changed)
Q_PROPERTY(bool data READ fjb2_dio_4 NOTIFY fjb2_dio_4Changed)
Q_PROPERTY(bool data READ fjb2_dio_5 NOTIFY fjb2_dio_5Changed)
Q_PROPERTY(bool data READ fjb2_dio_6 NOTIFY fjb2_dio_6Changed)
Q_PROPERTY(bool data READ fjb2_dio_7 NOTIFY fjb2_dio_7Changed)
Q_PROPERTY(unsigned int data READ fjb2_pwm_0 NOTIFY fjb2_pwm_0Changed)
Q_PROPERTY(unsigned int data READ fjb2_pwm_1 NOTIFY fjb2_pwm_1Changed)
Q_PROPERTY(unsigned int data READ fjb2_pwm_2 NOTIFY fjb2_pwm_2Changed)
Q_PROPERTY(unsigned int data READ fjb2_pwm_3 NOTIFY fjb2_pwm_3Changed)
Q_PROPERTY(unsigned int data READ fjb2_pwm_4 NOTIFY fjb2_pwm_4Changed)
Q_PROPERTY(unsigned int data READ fjb2_pwm_5 NOTIFY fjb2_pwm_5Changed)
Q_PROPERTY(unsigned int data READ fjb2_pwm_6 NOTIFY fjb2_pwm_6Changed)
Q_PROPERTY(unsigned int data READ fjb2_pwm_7 NOTIFY fjb2_pwm_7Changed)
Q_PROPERTY(unsigned int data READ fjb2_ai_0 NOTIFY fjb2_ai_0Changed)
Q_PROPERTY(unsigned int data READ fjb2_ai_1 NOTIFY fjb2_ai_1Changed)
Q_PROPERTY(unsigned int data READ fjb2_ai_2 NOTIFY fjb2_ai_2Changed)
Q_PROPERTY(unsigned int data READ fjb2_ai_3 NOTIFY fjb2_ai_3Changed)
Q_PROPERTY(unsigned int data READ fjb2_ai_4 NOTIFY fjb2_ai_4Changed)
Q_PROPERTY(unsigned int data READ fjb2_ai_5 NOTIFY fjb2_ai_5Changed)
Q_PROPERTY(unsigned int data READ fjb2_ai_6 NOTIFY fjb2_ai_6Changed)
Q_PROPERTY(unsigned int data READ fjb2_ai_7 NOTIFY fjb2_ai_7Changed)
Q_PROPERTY(bool data READ cjb1_di_0 NOTIFY cjb1_di_0Changed)
Q_PROPERTY(bool data READ cjb1_di_1 NOTIFY cjb1_di_1Changed)
Q_PROPERTY(bool data READ cjb1_di_2 NOTIFY cjb1_di_2Changed)
Q_PROPERTY(bool data READ cjb1_di_3 NOTIFY cjb1_di_3Changed)
Q_PROPERTY(bool data READ cjb1_do_0 NOTIFY cjb1_do_0Changed)
Q_PROPERTY(bool data READ cjb1_do_1 NOTIFY cjb1_do_1Changed)
Q_PROPERTY(bool data READ cjb1_do_2 NOTIFY cjb1_do_2Changed)
Q_PROPERTY(bool data READ cjb1_do_3 NOTIFY cjb1_do_3Changed)
Q_PROPERTY(bool data READ cjb1_do_4 NOTIFY cjb1_do_4Changed)
Q_PROPERTY(bool data READ cjb1_do_5 NOTIFY cjb1_do_5Changed)
Q_PROPERTY(bool data READ cjb1_do_6 NOTIFY cjb1_do_6Changed)
Q_PROPERTY(bool data READ cjb1_do_7 NOTIFY cjb1_do_7Changed)
Q_PROPERTY(bool data READ cjb1_do_8 NOTIFY cjb1_do_8Changed)
Q_PROPERTY(bool data READ cjb1_do_9 NOTIFY cjb1_do_9Changed)
Q_PROPERTY(bool data READ cjb1_do_10 NOTIFY cjb1_do_10Changed)
Q_PROPERTY(bool data READ cjb1_do_11 NOTIFY cjb1_do_11Changed)
Q_PROPERTY(bool data READ cjb1_dio_0 NOTIFY cjb1_dio_0Changed)
Q_PROPERTY(bool data READ cjb1_dio_1 NOTIFY cjb1_dio_1Changed)
Q_PROPERTY(bool data READ cjb1_dio_2 NOTIFY cjb1_dio_2Changed)

Q_PROPERTY(bool data READ cjb1_dio_3 NOTIFY cjb1_dio_3Changed)
Q_PROPERTY(bool data READ cjb1_dio_4 NOTIFY cjb1_dio_4Changed)
Q_PROPERTY(bool data READ cjb1_dio_5 NOTIFY cjb1_dio_5Changed)
Q_PROPERTY(bool data READ cjb1_dio_6 NOTIFY cjb1_dio_6Changed)
Q_PROPERTY(bool data READ cjb1_dio_7 NOTIFY cjb1_dio_7Changed)
Q_PROPERTY(unsigned int data READ cjb1_ai_0 NOTIFY cjb1_ai_0Changed)
Q_PROPERTY(unsigned int data READ cjb1_ai_1 NOTIFY cjb1_ai_1Changed)
Q_PROPERTY(unsigned int data READ cjb1_ai_2 NOTIFY cjb1_ai_2Changed)
Q_PROPERTY(unsigned int data READ cjb1_ai_3 NOTIFY cjb1_ai_3Changed)
Q_PROPERTY(unsigned int data READ cjb1_ai_4 NOTIFY cjb1_ai_4Changed)
Q_PROPERTY(unsigned int data READ cjb1_ai_5 NOTIFY cjb1_ai_5Changed)
Q_PROPERTY(unsigned int data READ cjb1_ai_6 NOTIFY cjb1_ai_6Changed)
Q_PROPERTY(unsigned int data READ cjb1_ai_7 NOTIFY cjb1_ai_7Changed)
Q_PROPERTY(bool data READ cjb2_do_0 NOTIFY cjb2_do_0Changed)
Q_PROPERTY(bool data READ cjb2_do_1 NOTIFY cjb2_do_1Changed)
Q_PROPERTY(bool data READ cjb2_do_2 NOTIFY cjb2_do_2Changed)
Q_PROPERTY(bool data READ cjb2_do_3 NOTIFY cjb2_do_3Changed)
Q_PROPERTY(bool data READ cjb2_dio_0 NOTIFY cjb2_dio_0Changed)
Q_PROPERTY(bool data READ cjb2_dio_1 NOTIFY cjb2_dio_1Changed)
Q_PROPERTY(bool data READ cjb2_dio_2 NOTIFY cjb2_dio_2Changed)
Q_PROPERTY(bool data READ cjb2_dio_3 NOTIFY cjb2_dio_3Changed)
Q_PROPERTY(bool data READ cjb2_di_0 NOTIFY cjb2_di_0Changed)
Q_PROPERTY(bool data READ cjb2_di_1 NOTIFY cjb2_di_1Changed)
Q_PROPERTY(bool data READ cjb2_di_2 NOTIFY cjb2_di_2Changed)
Q_PROPERTY(bool data READ cjb2_di_3 NOTIFY cjb2_di_3Changed)
Q_PROPERTY(bool data READ cjb2_dio_4 NOTIFY cjb2_dio_4Changed)
Q_PROPERTY(bool data READ cjb2_dio_5 NOTIFY cjb2_dio_5Changed)
Q_PROPERTY(bool data READ cjb2_dio_6 NOTIFY cjb2_dio_6Changed)
Q_PROPERTY(bool data READ cjb2_dio_7 NOTIFY cjb2_dio_7Changed)
Q_PROPERTY(unsigned int data READ cjb2_pwm_0 NOTIFY cjb2_pwm_0Changed)
Q_PROPERTY(unsigned int data READ cjb2_pwm_1 NOTIFY cjb2_pwm_1Changed)
Q_PROPERTY(unsigned int data READ cjb2_pwm_2 NOTIFY cjb2_pwm_2Changed)
Q_PROPERTY(unsigned int data READ cjb2_pwm_3 NOTIFY cjb2_pwm_3Changed)
Q_PROPERTY(unsigned int data READ cjb2_pwm_4 NOTIFY cjb2_pwm_4Changed)
Q_PROPERTY(unsigned int data READ cjb2_pwm_5 NOTIFY cjb2_pwm_5Changed)
Q_PROPERTY(unsigned int data READ cjb2_pwm_6 NOTIFY cjb2_pwm_6Changed)
Q_PROPERTY(unsigned int data READ cjb2_pwm_7 NOTIFY cjb2_pwm_7Changed)
Q_PROPERTY(unsigned int data READ cjb2_ai_0 NOTIFY cjb2_ai_0Changed)
Q_PROPERTY(unsigned int data READ cjb2_ai_1 NOTIFY cjb2_ai_1Changed)
Q_PROPERTY(unsigned int data READ cjb2_ai_2 NOTIFY cjb2_ai_2Changed)
Q_PROPERTY(unsigned int data READ cjb2_ai_3 NOTIFY cjb2_ai_3Changed)
Q_PROPERTY(unsigned int data READ cjb2_ai_4 NOTIFY cjb2_ai_4Changed)

Q_PROPERTY(unsigned int data READ cjb2_ai_5 NOTIFY cjb2_ai_5Changed)
Q_PROPERTY(unsigned int data READ cjb2_ai_6 NOTIFY cjb2_ai_6Changed)
Q_PROPERTY(unsigned int data READ cjb2_ai_7 NOTIFY cjb2_ai_7Changed)
Q_PROPERTY(bool data READ rjb1_di_0 NOTIFY rjb1_di_0Changed)
Q_PROPERTY(bool data READ rjb1_di_1 NOTIFY rjb1_di_1Changed)
Q_PROPERTY(bool data READ rjb1_di_2 NOTIFY rjb1_di_2Changed)
Q_PROPERTY(bool data READ rjb1_di_3 NOTIFY rjb1_di_3Changed)
Q_PROPERTY(bool data READ rjb1_do_0 NOTIFY rjb1_do_0Changed)
Q_PROPERTY(bool data READ rjb1_do_1 NOTIFY rjb1_do_1Changed)
Q_PROPERTY(bool data READ rjb1_do_2 NOTIFY rjb1_do_2Changed)
Q_PROPERTY(bool data READ rjb1_do_3 NOTIFY rjb1_do_3Changed)
Q_PROPERTY(bool data READ rjb1_do_4 NOTIFY rjb1_do_4Changed)
Q_PROPERTY(bool data READ rjb1_do_5 NOTIFY rjb1_do_5Changed)
Q_PROPERTY(bool data READ rjb1_do_6 NOTIFY rjb1_do_6Changed)
Q_PROPERTY(bool data READ rjb1_do_7 NOTIFY rjb1_do_7Changed)
Q_PROPERTY(bool data READ rjb1_do_8 NOTIFY rjb1_do_8Changed)
Q_PROPERTY(bool data READ rjb1_do_9 NOTIFY rjb1_do_9Changed)
Q_PROPERTY(bool data READ rjb1_do_10 NOTIFY rjb1_do_10Changed)
Q_PROPERTY(bool data READ rjb1_do_11 NOTIFY rjb1_do_11Changed)
Q_PROPERTY(bool data READ rjb1_dio_0 NOTIFY rjb1_dio_0Changed)
Q_PROPERTY(bool data READ rjb1_dio_1 NOTIFY rjb1_dio_1Changed)
Q_PROPERTY(bool data READ rjb1_dio_2 NOTIFY rjb1_dio_2Changed)
Q_PROPERTY(bool data READ rjb1_dio_3 NOTIFY rjb1_dio_3Changed)
Q_PROPERTY(bool data READ rjb1_dio_4 NOTIFY rjb1_dio_4Changed)
Q_PROPERTY(bool data READ rjb1_dio_5 NOTIFY rjb1_dio_5Changed)
Q_PROPERTY(bool data READ rjb1_dio_6 NOTIFY rjb1_dio_6Changed)
Q_PROPERTY(bool data READ rjb1_dio_7 NOTIFY rjb1_dio_7Changed)
Q_PROPERTY(unsigned int data READ rjb1_ai_0 NOTIFY rjb1_ai_0Changed)
Q_PROPERTY(unsigned int data READ rjb1_ai_1 NOTIFY rjb1_ai_1Changed)
Q_PROPERTY(unsigned int data READ rjb1_ai_2 NOTIFY rjb1_ai_2Changed)
Q_PROPERTY(unsigned int data READ rjb1_ai_3 NOTIFY rjb1_ai_3Changed)
Q_PROPERTY(unsigned int data READ rjb1_ai_4 NOTIFY rjb1_ai_4Changed)
Q_PROPERTY(unsigned int data READ rjb1_ai_5 NOTIFY rjb1_ai_5Changed)
Q_PROPERTY(unsigned int data READ rjb1_ai_6 NOTIFY rjb1_ai_6Changed)
Q_PROPERTY(unsigned int data READ rjb1_ai_7 NOTIFY rjb1_ai_7Changed)
Q_PROPERTY(bool data READ rjb2_do_0 NOTIFY rjb2_do_0Changed)
Q_PROPERTY(bool data READ rjb2_do_1 NOTIFY rjb2_do_1Changed)
Q_PROPERTY(bool data READ rjb2_do_2 NOTIFY rjb2_do_2Changed)
Q_PROPERTY(bool data READ rjb2_do_3 NOTIFY rjb2_do_3Changed)
Q_PROPERTY(bool data READ rjb2_dio_0 NOTIFY rjb2_dio_0Changed)
Q_PROPERTY(bool data READ rjb2_dio_1 NOTIFY rjb2_dio_1Changed)
Q_PROPERTY(bool data READ rjb2_dio_2 NOTIFY rjb2_dio_2Changed)

Q_PROPERTY(bool data READ rjb2_dio_3 NOTIFY rjb2_dio_3Changed)
Q_PROPERTY(bool data READ rjb2_di_0 NOTIFY rjb2_di_0Changed)
Q_PROPERTY(bool data READ rjb2_di_1 NOTIFY rjb2_di_1Changed)
Q_PROPERTY(bool data READ rjb2_di_2 NOTIFY rjb2_di_2Changed)
Q_PROPERTY(bool data READ rjb2_di_3 NOTIFY rjb2_di_3Changed)
Q_PROPERTY(bool data READ rjb2_dio_4 NOTIFY rjb2_dio_4Changed)
Q_PROPERTY(bool data READ rjb2_dio_5 NOTIFY rjb2_dio_5Changed)
Q_PROPERTY(bool data READ rjb2_dio_6 NOTIFY rjb2_dio_6Changed)
Q_PROPERTY(bool data READ rjb2_dio_7 NOTIFY rjb2_dio_7Changed)
Q_PROPERTY(unsigned int data READ rjb2_pwm_0 NOTIFY rjb2_pwm_0Changed)
Q_PROPERTY(unsigned int data READ rjb2_pwm_1 NOTIFY rjb2_pwm_1Changed)
Q_PROPERTY(unsigned int data READ rjb2_pwm_2 NOTIFY rjb2_pwm_2Changed)
Q_PROPERTY(unsigned int data READ rjb2_pwm_3 NOTIFY rjb2_pwm_3Changed)
Q_PROPERTY(unsigned int data READ rjb2_pwm_4 NOTIFY rjb2_pwm_4Changed)
Q_PROPERTY(unsigned int data READ rjb2_pwm_5 NOTIFY rjb2_pwm_5Changed)
Q_PROPERTY(unsigned int data READ rjb2_pwm_6 NOTIFY rjb2_pwm_6Changed)
Q_PROPERTY(unsigned int data READ rjb2_pwm_7 NOTIFY rjb2_pwm_7Changed)
Q_PROPERTY(unsigned int data READ rjb2_ai_0 NOTIFY rjb2_ai_0Changed)
Q_PROPERTY(unsigned int data READ rjb2_ai_1 NOTIFY rjb2_ai_1Changed)
Q_PROPERTY(unsigned int data READ rjb2_ai_2 NOTIFY rjb2_ai_2Changed)
Q_PROPERTY(unsigned int data READ rjb2_ai_3 NOTIFY rjb2_ai_3Changed)
Q_PROPERTY(unsigned int data READ rjb2_ai_4 NOTIFY rjb2_ai_4Changed)
Q_PROPERTY(unsigned int data READ rjb2_ai_5 NOTIFY rjb2_ai_5Changed)
Q_PROPERTY(unsigned int data READ rjb2_ai_6 NOTIFY rjb2_ai_6Changed)
Q_PROPERTY(unsigned int data READ rjb2_ai_7 NOTIFY rjb2_ai_7Changed)
Q_PROPERTY(bool data READ epec_error_bit_0 NOTIFY epec_error_bit_0Changed)
Q_PROPERTY(bool data READ epec_error_bit_1 NOTIFY epec_error_bit_1Changed)
Q_PROPERTY(bool data READ epec_error_bit_2 NOTIFY epec_error_bit_2Changed)
Q_PROPERTY(bool data READ epec_error_bit_3 NOTIFY epec_error_bit_3Changed)
Q_PROPERTY(bool data READ epec_error_bit_4 NOTIFY epec_error_bit_4Changed)
Q_PROPERTY(bool data READ epec_error_bit_5 NOTIFY epec_error_bit_5Changed)
Q_PROPERTY(unsigned int data READ epec_error_byte NOTIFY epec_error_byteChanged)

TPDO-viestit

IVI_NAME	PDO_INDEX	NODE_ID	COBID	TYPE	LENGTH_BIT	BYTE_INDEX	BIT_INDEX
Front Glass Heater on/off	0	80	464	bool	1		0
Rear Glass Heater on/off	0	80	464	bool	1		1
AC on/off	0	80	464	bool	1		2
Steering Motor on/off	0	80	464	bool	1		3
AC temp	1	80	720	unsigned char	8	0	
AC fan	1	80	720	unsigned char	8	1	
Left Mirror Up on/off	2	80	976	bool	1		0
Left Mirror Down on/off	2	80	976	bool	1		1
Left Mirror Left on/off	2	80	976	bool	1		2
Left Mirror Right on/off	2	80	976	bool	1		3
Right Mirror Up on/off	2	80	976	bool	1		4
Right Mirror Down on/off	2	80	976	bool	1		5
Right Mirror Left on/off	2	80	976	bool	1		6
Right Mirror Right on/off	2	80	976	bool	1		7
Headlights on/off	2	80	976	bool	1		8
Park on/off	2	80	976	bool	1		9
Front Fog Lights on/off	2	80	976	bool	1		11
Rear Fog Lights on/off	2	80	976	bool	1		12
Doors Lock on/off	2	80	976	bool	1		13
Interior Lights on/off	2	80	976	bool	1		14
Interior Lights Brightness	3	80	1232	unsigned char	8	0	
Interior Lights Color R	3	80	1232	unsigned char	8	1	
Interior Lights Color G	3	80	1232	unsigned char	8	2	

Interior Lights Color B	3	80	1232	unsigned char	8	3	
Suspension Level	3	80	1232	unsigned char	8	4	
Steering Motor Speed	3	80	1232	unsigned char	8	5	

RPDO-viestit

JB_PIN	PDO_INDEX	NODE_ID	CO-BID	TYPE	LENGTH_BIT	BY-TE_INDEX	BIT_INDEX	CON-VERT_FUNC-TION	BIT_SHIFT_RIGHT	BIT_SHIFT_LEFT
FJB1_DI_0	0	69	453	bool	1		0			
FJB1_DI_1	0	69	453	bool	1		1			
FJB1_DI_2	0	69	453	bool	1		2			
FJB1_DI_3	0	69	453	bool	1		3			
FJB1_DO_0	1	69	581	bool	1		0			
FJB1_DO_1	1	69	581	bool	1		1			
FJB1_DO_2	1	69	581	bool	1		2			
FJB1_DO_3	1	69	581	bool	1		3			
FJB1_DO_4	1	69	581	bool	1		4			
FJB1_DO_5	1	69	581	bool	1		5			
FJB1_DO_6	1	69	581	bool	1		6			
FJB1_DO_7	1	69	581	bool	1		7			
FJB1_DO_8	1	69	581	bool	1		8			
FJB1_DO_9	1	69	581	bool	1		9			
FJB1_DO_10	1	69	581	bool	1		10			
FJB1_DO_11	1	69	581	bool	1		11			
FJB1_DIO_0	1	69	581	bool	1		12			
FJB1_DIO_1	1	69	581	bool	1		13			
FJB1_DIO_2	1	69	581	bool	1		14			

FJB1_DIO_3	1	69	581	bool	1		15			
FJB1_DIO_4	1	69	581	bool	1		16			
FJB1_DIO_5	1	69	581	bool	1		17			
FJB1_DIO_6	1	69	581	bool	1		18			
FJB1_DIO_7	1	69	581	bool	1		19			
FJB1_AI_0	2	69	709	unsig sig-ned char	8	0		Conver-tAna-log8Bit		
FJB1_AI_1	2	69	709	unsig sig-ned char	8	1		Conver-tAna-log8Bit		
FJB1_AI_2	2	69	709	unsig sig-ned char	8	2		Conver-tAna-log8Bit		
FJB1_AI_3	2	69	709	unsig sig-ned char	8	3		Conver-tAna-log8Bit		
FJB1_AI_4	2	69	709	unsig sig-ned char	8	4		Conver-tAna-log8Bit		
FJB1_AI_5	2	69	709	unsig sig-ned char	8	5		Conver-tAna-log8Bit		
FJB1_AI_6	2	69	709	unsig sig-ned char	8	6		Conver-tAna-log8Bit		
FJB1_AI_7	2	69	709	unsig sig-ned char	8	7		Conver-tAna-log8Bit		
FJB2_DO_0	3	68	580	bool	1		8			
FJB2_DO_1	3	68	580	bool	1		9			

FJB2_DO_2	3	68	580	bool	1		10			
FJB2_DO_3	3	68	580	bool	1		11			
FJB2_DIO_0	3	68	580	bool	1		12			
FJB2_DIO_1	3	68	580	bool	1		13			
FJB2_DIO_2	3	68	580	bool	1		14			
FJB2_DIO_3	3	68	580	bool	1		15			
FJB2_DI_0	4	68	452	bool	1		8			
FJB2_DI_1	4	68	452	bool	1		9			
FJB2_DI_2	4	68	452	bool	1		10			
FJB2_DI_3	4	68	452	bool	1		11			
FJB2_DIO_4	4	68	452	bool	1		12			
FJB2_DIO_5	4	68	452	bool	1		13			
FJB2_DIO_6	4	68	452	bool	1		14			
FJB2_DIO_7	4	68	452	bool	1		15			
FJB2_PWM_0	5	68	836	unsig sig-ned short	16	0				5
FJB2_PWM_1	5	68	836	unsig sig-ned short	16	2				5
FJB2_PWM_2	5	68	836	unsig sig-ned short	16	4				5
FJB2_PWM_3	5	68	836	unsig sig-ned short	16	6				5

FJB2_PWM_4	6	68	1092	unsig sig-ned short	16	0				5
FJB2_PWM_5	6	68	1092	unsig sig-ned short	16	2				5
FJB2_PWM_6	6	68	1092	unsig sig-ned short	16	4				5
FJB2_PWM_7	6	68	1092	unsig sig-ned short	16	6				5
FJB2_AI_0	7	68	708	unsig sig-ned short	16	0			5	
FJB2_AI_1	7	68	708	unsig sig-ned short	16	2			5	
FJB2_AI_2	7	68	708	unsig sig-ned short	16	4			5	
FJB2_AI_3	7	68	708	unsig sig-ned short	16	6			5	
FJB2_AI_4	8	68	964	unsig sig-ned short	16	0			5	
FJB2_AI_5	8	68	964	unsig sig-ned short	16	2			5	
FJB2_AI_6	8	68	964	unsig sig-ned short	16	4			5	
FJB2_AI_7	8	68	964	unsig sig-ned short	16	6			5	

CJB1_DI_0	9	64	448	bool	1		0			
CJB1_DI_1	9	64	448	bool	1		1			
CJB1_DI_2	9	64	448	bool	1		2			
CJB1_DI_3	9	64	448	bool	1		3			
CJB1_DO_0	10	64	576	bool	1		0			
CJB1_DO_1	10	64	576	bool	1		1			
CJB1_DO_2	10	64	576	bool	1		2			
CJB1_DO_3	10	64	576	bool	1		3			
CJB1_DO_4	10	64	576	bool	1		4			
CJB1_DO_5	10	64	576	bool	1		5			
CJB1_DO_6	10	64	576	bool	1		6			
CJB1_DO_7	10	64	576	bool	1		7			
CJB1_DO_8	10	64	576	bool	1		8			
CJB1_DO_9	10	64	576	bool	1		9			
CJB1_DO_10	10	64	576	bool	1		10			
CJB1_DO_11	10	64	576	bool	1		11			
CJB1_DIO_0	10	64	576	bool	1		12			
CJB1_DIO_1	10	64	576	bool	1		13			
CJB1_DIO_2	10	64	576	bool	1		14			
CJB1_DIO_3	10	64	576	bool	1		15			
CJB1_DIO_4	10	64	576	bool	1		16			
CJB1_DIO_5	10	64	576	bool	1		17			

CJB1_DIO_6	10	64	576	bool	1		18			
CJB1_DIO_7	10	64	576	bool	1		19			
CJB1_AI_0	11	64	704	unsig sig-ned char	8	0		Conver-tAna-log8Bit		
CJB1_AI_1	11	64	704	unsig sig-ned char	8	1		Conver-tAna-log8Bit		
CJB1_AI_2	11	64	704	unsig sig-ned char	8	2		Conver-tAna-log8Bit		
CJB1_AI_3	11	64	704	unsig sig-ned char	8	3		Conver-tAna-log8Bit		
CJB1_AI_4	11	64	704	unsig sig-ned char	8	4		Conver-tAna-log8Bit		
CJB1_AI_5	11	64	704	unsig sig-ned char	8	5		Conver-tAna-log8Bit		
CJB1_AI_6	11	64	704	unsig sig-ned char	8	6		Conver-tAna-log8Bit		
CJB1_AI_7	11	64	704	unsig sig-ned char	8	7		Conver-tAna-log8Bit		
CJB2_DO_0	12	65	577	bool	1		8			
CJB2_DO_1	12	65	577	bool	1		9			
CJB2_DO_2	12	65	577	bool	1		10			
CJB2_DO_3	12	65	577	bool	1		11			
CJB2_DIO_0	12	65	577	bool	1		12			

CJB2_DIO_1	12	65	577	bool	1		13			
CJB2_DIO_2	12	65	577	bool	1		14			
CJB2_DIO_3	12	65	577	bool	1		15			
CJB2_DI_0	13	65	449	bool	1		8			
CJB2_DI_1	13	65	449	bool	1		9			
CJB2_DI_2	13	65	449	bool	1		10			
CJB2_DI_3	13	65	449	bool	1		11			
CJB2_DIO_4	13	65	449	bool	1		15			
CJB2_DIO_5	13	65	449	bool	1		14			
CJB2_DIO_6	13	65	449	bool	1		13			
CJB2_DIO_7	13	65	449	bool	1		12			
CJB2_PWM_0	14	65	833	unsig sig-ned short	16	0				5
CJB2_PWM_1	14	65	833	unsig sig-ned short	16	2				5
CJB2_PWM_2	14	65	833	unsig sig-ned short	16	4				5
CJB2_PWM_3	14	65	833	unsig sig-ned short	16	6				5
CJB2_PWM_4	15	65	1089	unsig sig-ned short	16	0				5
CJB2_PWM_5	15	65	1089	unsig sig-ned short	16	2				5

CJB2_PWM_6	15	65	1089	unsig sig-ned short	16	4				5
CJB2_PWM_7	15	65	1089	unsig sig-ned short	16	6				5
CJB2_AI_0	16	65	705	unsig sig-ned short	16	0			5	
CJB2_AI_1	16	65	705	unsig sig-ned short	16	2			5	
CJB2_AI_2	16	65	705	unsig sig-ned short	16	4			5	
CJB2_AI_3	16	65	705	unsig sig-ned short	16	6			5	
CJB2_AI_4	17	65	961	unsig sig-ned short	16	0			5	
CJB2_AI_5	17	65	961	unsig sig-ned short	16	2			5	
CJB2_AI_6	17	65	961	unsig sig-ned short	16	4			5	
CJB2_AI_7	17	65	961	unsig sig-ned short	16	6			5	
RJB1_DI_0	18	17	401	bool	1		0			
RJB1_DI_1	18	17	401	bool	1		1			
RJB1_DI_2	18	17	401	bool	1		2			
RJB1_DI_3	18	17	401	bool	1		3			

RJB1_DO_0	19	17	529	bool	1		0			
RJB1_DO_1	19	17	529	bool	1		1			
RJB1_DO_2	19	17	529	bool	1		2			
RJB1_DO_3	19	17	529	bool	1		3			
RJB1_DO_4	19	17	529	bool	1		4			
RJB1_DO_5	19	17	529	bool	1		5			
RJB1_DO_6	19	17	529	bool	1		6			
RJB1_DO_7	19	17	529	bool	1		7			
RJB1_DO_8	19	17	529	bool	1		8			
RJB1_DO_9	19	17	529	bool	1		9			
RJB1_DO_10	19	17	529	bool	1		10			
RJB1_DO_11	19	17	529	bool	1		11			
RJB1_DIO_0	19	17	529	bool	1		12			
RJB1_DIO_1	19	17	529	bool	1		13			
RJB1_DIO_2	19	17	529	bool	1		14			
RJB1_DIO_3	19	17	529	bool	1		15			
RJB1_DIO_4	19	17	529	bool	1		16			
RJB1_DIO_5	19	17	529	bool	1		17			
RJB1_DIO_6	19	17	529	bool	1		18			
RJB1_DIO_7	19	17	529	bool	1		19			
RJB1_AI_0	20	17	657	unsig sig-ned char	8	0		Conver-tAna-log8Bit		

RJB1_AI_1	20	17	657	unsig sig-ned char	8	1		Conver-tAna-log8Bit		
RJB1_AI_2	20	17	657	unsig sig-ned char	8	2		Conver-tAna-log8Bit		
RJB1_AI_3	20	17	657	unsig sig-ned char	8	3		Conver-tAna-log8Bit		
RJB1_AI_4	20	17	657	unsig sig-ned char	8	4		Conver-tAna-log8Bit		
RJB1_AI_5	20	17	657	unsig sig-ned char	8	5		Conver-tAna-log8Bit		
RJB1_AI_6	20	17	657	unsig sig-ned char	8	6		Conver-tAna-log8Bit		
RJB1_AI_7	20	17	657	unsig sig-ned char	8	7		Conver-tAna-log8Bit		
RJB2_DO_0	21	16	528	bool	1		8			
RJB2_DO_1	21	16	528	bool	1		9			
RJB2_DO_2	21	16	528	bool	1		10			
RJB2_DO_3	21	16	528	bool	1		11			
RJB2_DIO_0	21	16	528	bool	1		12			
RJB2_DIO_1	21	16	528	bool	1		13			
RJB2_DIO_2	21	16	528	bool	1		14			
RJB2_DIO_3	21	16	528	bool	1		15			
RJB2_DI_0	22	16	400	bool	1		8			

RJB2_DI_1	22	16	400	bool	1		9			
RJB2_DI_2	22	16	400	bool	1		10			
RJB2_DI_3	22	16	400	bool	1		11			
RJB2_DIO_4	22	16	400	bool	1		12			
RJB2_DIO_5	22	16	400	bool	1		13			
RJB2_DIO_6	22	16	400	bool	1		14			
RJB2_DIO_7	22	16	400	bool	1		15			
RJB2_PWM_0	23	16	784	unsig sig-ned short	16	0				5
RJB2_PWM_1	23	16	784	unsig sig-ned short	16	2				5
RJB2_PWM_2	23	16	784	unsig sig-ned short	16	4				5
RJB2_PWM_3	23	16	784	unsig sig-ned short	16	6				5
RJB2_PWM_4	24	16	1040	unsig sig-ned short	16	0				5
RJB2_PWM_5	24	16	1040	unsig sig-ned short	16	2				5
RJB2_PWM_6	24	16	1040	unsig sig-ned short	16	4				5
RJB2_PWM_7	24	16	1040	unsig sig-ned short	16	6				5

RJB2_AI_0	25	16	656	unsig sig-ned short	16	0			5	
RJB2_AI_1	25	16	656	unsig sig-ned short	16	2			5	
RJB2_AI_2	25	16	656	unsig sig-ned short	16	4			5	
RJB2_AI_3	25	16	656	unsig sig-ned short	16	6			5	
RJB2_AI_4	26	16	912	unsig sig-ned short	16	0			5	
RJB2_AI_5	26	16	912	unsig sig-ned short	16	2			5	
RJB2_AI_6	26	16	912	unsig sig-ned short	16	4			5	
RJB2_AI_7	26	16	912	unsig sig-ned short	16	6			5	
EPEC_ERROR_BIT_0	27	80	592	bool	1		0			
EPEC_ERROR_BIT_1	27	80	592	bool	1		1			
EPEC_ERROR_BIT_2	27	80	592	bool	1		2			
EPEC_ERROR_BIT_3	27	80	592	bool	1		3			
EPEC_ERROR_BIT_4	27	80	592	bool	1		4			

EPEC_ERROR_BIT_5	27	80	592	bool	1		5			
EPEC_ERROR_BYTE	28	80	848	unsig sig-ned char	8	0				